NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

THE VIABILITY OF A DTN SYSTEM FOR CURRENT MILITARY APPLICATION

by

Todd J. Sehl

March 2013

Thesis Advisor: Geoffrey G. Xie
Co-Advisor: Justin P. Rohrer

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 2013	Master's Thesis

4. TITLE AND SUBTITLE
THE VIABILITY OF A DTN SYSTEM FOR CURRENT MILITARY APPLICATION

5. FUNDING NUMBERS

6. AUTHOR(S) Todd J. Sehl

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____ .

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

With DTN technology in development we see the DARPA Disruption-Tolerant Networking program and the DTN Research Group making significant strides toward disruption-tolerant network infrastructure. Mobile ad-hoc networks are a topic of interest in the military today due to the flexibility of the network to expand and contract continuously and remain consistent in a highly changing environment. The primary research question in this thesis is the viability of the SPINDLE Disruption-Tolerant Networking software developed for field deployment in the United States Marine Corps. My research evaluates the usability of the BBN SPINDLE BPA for deployment. In this paper, I discuss what is required to learn, install, and configure the BBN software while evaluating how stable the software performs. It explores the question of if it is feasible to add an ICMP notification service for applications whose traffic has been diverted due to the DTN process. The tests conducted demonstrate two possible methods to use ICMP messages in a network to convey unique DTN messages to individual hosts. It demonstrates how a known ICMP message type can be utilized to carry message flags representing explicit network disruption notifications in applications designed to recognize them.

14. SUBJECT TERMS	15. NUMBER OF PAGES
DTN, Delay-Tolerant Network, Disruption-Tolerant Network, SPINDLE, Networks, BBN, Raytheon	109
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

THE VIABILITY OF A DTN SYSTEM FOR CURRENT MILITARY APPLICATION

Todd J. Sehl
Lieutenant Commander, United States Navy
B.A., Norwich University, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2013

Author: Todd J. Sehl

Approved by: Geoffrey G. Xie
Thesis Advisor

Justin P. Rohrer
Thesis Co-Advisor

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

With DTN technology in development we see the DARPA Disruption-Tolerant Networking program and the DTN Research Group making significant strides toward disruption-tolerant network infrastructure. Mobile ad-hoc networks are a topic of interest in the military today due to the flexibility of the network to expand and contract continuously and remain consistent in a highly changing environment. The primary research question in this thesis is the viability of the SPINDLE Disruption-Tolerant Networking software developed for field deployment in the United States Marine Corps. My research evaluates the usability of the BBN SPINDLE BPA for deployment. In this paper, I discuss what is required to learn, install, and configure the BBN software while evaluating how stable the software performs. It explores the question of if it is feasible to add an ICMP notification service for applications whose traffic has been diverted due to the DTN process. The tests conducted demonstrate two possible methods to use ICMP messages in a network to convey unique DTN messages to individual hosts. It demonstrates how a known ICMP message type can be utilized to carry message flags representing explicit network disruption notifications in applications designed to recognize them.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

A/M	Application/Middleware
AA	Average Availability
APLS	Anxiety-Prone Link State
CLA	Convergence Layer Adapter
BGP	Border Gateway Protocol
BPA	Bundle Protocol Agent
BSP	Bundle Security Protocol
CLA	Convergence Layer Adapter
CR	Connected Routing
DARPA	Defense Advanced Research Projects Agency
DP	Decision Plane
DR	Disrupted Routing
DS	Data Store
DTN	Disruption-Tolerant Network
DTNRG	Delay-Tolerant Networking Research Group
EID	Endpoint Identifier
GRB	Gamma Ray Burst
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IGRP	Interior Gateway Routing Protocol
IP	Internet Protocol
IPN	Interplanetary Internet
KB	Knowledge Base
MANET	Mobile Adhoc Network
MTU	Maximum Transmission Unit
ND	Neighbor Discovery
NPS	Naval Postgraduate School
OPAL	Open Phone Abstraction Library
OSPF	Open Shortest Path First
PDU	Protocol Data Unit

PREP	Prioritized Epidemic
PTLib	Portable Tools Library
RIP	Routing Information Protocol
SCP	Shortest Cost Path
SDNV	Self-Delimiting Numeric Values
SIP	Session Initiation Protocol
SPINDLE	Survivable Policy-Influenced Networking: Disruption-tolerance through Learning and Evolution
SSH	Secure Shell
SSP	Scheme-Specific Part
TCP	Transmission Control Program
TMRG	Transport Modeling Research Group
URL	Universal Resource Locator
VANET	Vehicle Adhoc Network
VoIP	Voice over IP
XML	Extensible Markup Language

ACKNOWLEDGMENTS

I would like to extend my thanks and appreciation to several people for their assistance in making this thesis a valuable learning experience.

I would like to extend a personal thanks to my primary thesis advisor, Dr. Geoffrey Xie, whose patience and wisdom led to the successful completion of this paper. I would also like to thank my thesis co-advisor, Justin Rohrer, for his insight into the programming and Linux world that I do not have.

I was also fortunate to be buoyed by the support, encouragement, and pride of my wife, my kids, my parents, my sisters and brother, their families, as well as my loving second family that I inherited when I married my wife.

I would like to reiterate the deep, heartfelt, and special thanks I extended to my wife, Ashley, son, Ian, and daughter, Gabrielle. They have all been an exceptional support base. Ashley, thank you for your loving support and prayers, providing your love and understanding on a daily basis; you really bring out the best in me. Ian, thank you for helping to remind me that there is more to math than Automata. Gabrielle, thank you for keeping me young at heart and providing me with those daily reminders of what is really important in life. Without all of you, none of this could have been possible.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM

Delay- and Disruption-Tolerant Networking (DTN) technology has been in development for years, with the first documented use of the term delay-tolerant networking in 2002 by Kevin Fall when adapting interplanetary networking ideas to terrestrial networks [1]. Most recently in this area the Defense Advanced Research Project Agency (DARPA) Disruption-Tolerant Networking program and the Internet Research Task Force (IRTF) DTN Research Group made significant strides toward localized disruption-tolerant network hardware and software that extends the reach of an ad-hoc network infrastructure. Mobile ad-hoc networks are a topic of interest in the military today due to the flexibility of the network to expand and contract continuously and remain consistent in a highly changing environment. Deployed combat units require networks that are highly forgiving of infrastructure changes and frequent losses of connectivity. Current protocols and network hardware are designed not to accommodate long delays.

This thesis is an assessment of a current technology available to run disruption tolerant network applications. The scope of this thesis is to test the current release of disruption-tolerant network software, collecting the data and conducting a comparison to the advertised operation. This research paper will detail the testing of the advertised capabilities of a Raytheon/BBN DTN implementation in comparison to the published user manual. I will observe the performance of two distinct routing

topologies detailed in the user manual. I will also observe the performance of some applications offered by the DTN BPA.

The observations made here are of a current existing disruption-tolerant network technology developed by BBN Technologies to test protocols with applications operating in a test environment. With several platforms sending and receiving traffic through this network using the disruption-tolerant network software and protocols, we can observe traffic reports of data across the routers to analyze. I will observe and evaluate common data as well as introduce traffic for operational analysis. Once the lab testing has progressed to an acceptable point our hope is to deploy the equipment to a real-world environment and run similar tests for comparison data. The results from this testing will serve as feedback for current usage of SPINDLE software. This feedback will also assist in future design decisions about developing DTNs for use in real-world military deployment in mobile ad-hoc networks. It is also important to evaluate how feasible it is to alter the ICMP notification service for applications whose traffic has been diverted to the DTN process.

In the military today, we see the use of mobile devices proliferating throughout the daily operations of even the smallest unit. In fact, the small units deployed to the field seem to be in need of small compact mobile devices most of all. I have walked though my unit’s office while en route to accomplish a simple task and seen mobile phones on many of my co-workers’ desks. Waiting to be used or in use they are constantly probing for networks to

connect to. Mobile phones are the obvious example of this but there are several examples of that you may not even notice. If you simply look around you will see how much of our life is wireless. From wireless security cameras to wireless headphones to wireless routers, every aspect of our life offers a wireless option. Yet a person can wonder how connected we are really. All these devices are wireless but they require an access point that can be relied upon continuously. These devices do not communicate any data unless they have an established end-to-end link. This is the current standard of our wireless technology.

Current technology relies on strict standards and firm addressing of source and destination to establish a route for data transfer. With shortest path cost metrics defined through routing protocols such as OSPF, RIP, BGP, and IGRP. These protocols are excellent for wired networks, which know where each and every device can be found. This is accomplished through not physical locations but logical addresses assigned upon entering the network and renewed periodically. Once a device enters a wired network it is not expected to move out rapidly or lose connectivity frequently. If it does, the device will need to be re-authenticated each time.

Wireless devices require an access point to connect to the network. In 802.11, this is an access point to a physical network of routers and servers. For mobile phones and other devices that require cellular coverage connect to controllers via base stations designed to provide the mobile devices a connection to the Internet. Wireless protocols assign an IP and expect a certain degree of

compliance to the "remain in one location" theory as well. Otherwise the access point will lose contact with the device and de-authenticate it losing the ability to send and receive traffic. Even with mobile phones there is a handoff to another controller to maintain access to whatever resource the user is attempting to traffic data with. If there is a loss of connectivity then they need to start over. This creates a lot of overhead when you try to apply these protocols to a mobile network. Mobile networks need to be ad-hoc to accommodate the entering and leaving of devices so rapidly. VANETs are also a prime example of networks that rely on the fluidity of devices moving and need protocols to support this nature.

This is where DTNs become extremely sought after. DTNs are rapidly approaching real-world application needs. In the military, MANETs and VANETs are being studied increasingly due to usefulness in the field for all sorts of units. Raytheon BBN Technologies, funded by the DARPA DTNRG program, has developed software that establishes and maintains a network between devices that is disruption-tolerant utilizing protocols that operate differently than the common standards of Transmission Control Protocol (TCP)/Internet Protocol (IP) used today.

B. RESEARCH QUESTIONS

The primary research question for this thesis will be to evaluate the viability of the current release of the BBN SPINDLE Disruption Tolerant Networking software developed for field deployment in the United States Marine Corps (USMC). This thesis will also explore the questions:

- Is the BBN software easy to install and configure?
- Are there sufficient scenarios included in the tutorial?
- Is the code stable?
- Does the new version correct the bugs reported?

Using these questions this thesis will demonstrate the DTN software developed and provided to the USMC to enable access to information when stable end-to-end paths do not exist and network infrastructure access cannot be assured. Follow on research could be directed at testing and development of similar networking software that can enhance performance of ad-hoc networks in other Department of Defense branches.

C. STRUCTURE OF THIS THESIS

Including the introduction, this thesis is organized into six chapters. They outline the concept and testing done to the current version of SPINDLE software. While this research paper tests certain capabilities in the SPINDLE software future research is suggested to further DTN usage in military operations.

1. Chapter I: Introduction

This chapter is the introduction to this thesis detailing the problem seen, the research question and a brief description of each of the following chapters.

2. Chapter II: Background

This chapter is a condensed review of previous attempts at delay/disruption tolerant networks and current programs working on developing new technologies in the DTN area of research. It explains the difference between delay and disruption when referring to DTNs. There is an overview of the BBN SPINDLE Architecture. It will detail the make-up of the standard DTN bundles and how they compare to current TCP/IP packet data units. This chapter details the SPINDLE Disruption-Tolerant Networking System and explains the intended use. It will describe the technology innovations to include the routing algorithms, name-management architecture for DTNs, distributed caching, indexing, and retrieval approaches for disruption tolerant content-based (rather than locator-based) access to information, and a declarative knowledge-based approach that integrates routing, intentional naming, policy-based resource management, and content-based access to information.

3. Chapter III: Test Topologies and Methodology

This chapter provides a short description of the five different topologies that can be utilized in the BBN Disruption-Tolerant Network software. It will describe the tests conducted on the BBN SPINDLE software as well as the tests to be conducted on ICMP messages.

4. Chapter IV: BBN SPINDLE Software Testing Results

This chapter will discuss the testing and evaluation of the BBN SPINDLE BPA software. It will compare the advertised operation to actual operation in two separate topologies. It covers the installation, static routing

topology, epidemic routing topology, and review whether known errors were fixed.

5. Chapter V: Adding ICMP Recognition of DTN to Ekiga

This chapter will discuss the Ekiga Softphone and libraries used. It will explore its use in testing DTN software as well how I used C++ code to send and receive DTN special messages via ICMP packets. This software will be used in future testing and evaluation of the BBN SPINDLE BPA software. I detail the tests used to determine of ICMP is a feasible use for DTN unique messages in a network.

6. Chapter VI: Conclusion

A brief reiteration of the goals of the thesis and testing done to verify the BBN SPINDLE BPA software, with a condensed overview of the results. I discuss the feasibility of the BBN SPINDLE software for deployment and possible future testing for DTN software.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. HISTORY

Research into networks that can tolerate a delay in traffic due to some sort of medium extremities began early in the lifetime of computer technology. The United States' government found a need to overcome significant delays and packet corruption due to the multiple mediums between space and the surface of the earth. Long delays and timeouts experienced in transferring traffic to and from space made using normal network protocols and technology problematic. Through DARPA, the U.S. government offered funding to NASA, MITRE, and other research groups to develop a proposal for the Interplanetary Internet (based on early Interplanetary Network (IPN) design) hoping to solve some of these issues. The goal was to improve the networking with short-range manned missions to the moon and back. The Interplanetary Internet is made up of a computer network in space, consisting of a set of network nodes that can communicate with each other over long distances and tolerate large delays in traffic. The Interplanetary Internet is a store-and-forward *network of Internets* that is often disconnected, has a wireless backbone fraught with error-prone links and speed-of-light delays ranging from tens of minutes to even hours, even when there is a connection [2]. At the same time, as the research continued into advancing the interplanetary network technology, another form of network technology closer to home was being developed, delay tolerant networks. With this expansion into delay tolerant networks researchers also looked at disruption-

tolerant networks. Military organizations such as DARPA were interested in disruption-tolerant networks vice delay tolerant due to the differences in the nature of delay and disruption tolerant networks. Delay tolerant networks are designed to combat long latency while disruption-tolerant are designed to accommodate delay as well as other communications problems that may lead to disrupted traffic.

As early as the 1970s, advancements of computing encouraged researchers to develop technology to support routing between non-fixed locations of computers. The field did not see much advancement though until the 1990s when mobile devices became more popular. With wireless protocols in the interest, mobile ad-hoc networking (MANET) and vehicular ad-hoc networking (VANET) became areas of increasing interest.

B. DELAY-TOLERANCE VERSUS DISRUPTION-TOLERANCE

In 2002, researcher Kevin Fall began adapting the IPN technology with the intention to adapt it to terrestrial networks. The terms “delay-tolerant networking” and “DTN” were used for the first time in his paper “A Delay-Tolerant Network Architecture for Challenged Internets” published in 2003 [1]. With a growing interest in disruption-tolerant networks and the protocols that were being proposed, research expanded into disruption-tolerant networks as well. Even though both delay-tolerant networks and disruption-tolerant networks have both been referred to as DTNs there is a difference. In the delay-tolerant networks proposed by Kevin Fall, the purpose is to overcome the obstacles encountered by the extreme distances, in which long latency is expected as in IPNs. Delay-tolerant

networks are just a portion of the research on disruption-tolerant networks as a whole. Disruption-tolerant networks however deal with all the issues of intermittent communications problems, transmission limitations, and anomalies present either as a by-product of network traffic conditions, insufficient infrastructure, or intentional attack.

With mobile technology evolving and potential communication environments being developed, most mobile environments do not conform to the Internet's underlying assumptions of a guaranteed connection and a reliable path to the destination now. There are multiple categories of challenges to connected communication models. These include, but are not limited to, Intermittent Connectivity, Long or Variable Delay, Asymmetric Data Rates, and High Error Rates [3].

Some say, "There is nothing new under the sun" and I would be forced to believe them even in this new world of computer networks. One proposed way to achieve a DTN is to use store-and-forward message switching. A method that forwards the messages to each node and stores them until it is capable of forwarding it to the next hop en route to the destination. The pony express used this very same method with the rush to the next station and then waits for the next horse to arrive to send it on.

In today's active world, we are constantly on the move. We want networks that can keep up and technology that can make our travels easier. For example the development of MANETs and VANETs requires new technology and software that

is tolerant of unexpected loss in communication or intermittent connectivity. This is the case for interplanetary Internet as well as common terrestrial devices today. [3] Because this work consists of research on the SPINDLE technology developed by BBN, the term DTN will be used as a reference to disruption-tolerant networks.

C. SPINDLE ARCHITECTURE

Under the DARPA DTN program, BBN has developed the Survivable Policy-Influenced Networking: Disruption-tolerance through Learning and Evolution (SPINDLE) DTN system. SPINDLE DTN is a modular system designed to enable a common core to be used with multiple expansion modules as seen in Figure 1. SPINDLE builds upon the DTNRG reference implementation (DTN2), which can be downloaded from [4]. Each new module will provide services to the node it is implemented on. This modular design allows for researchers to develop new algorithms expanding on the functions available to the common core of the DTNRG standards and software [5]. Each component of SPINDLE is attributed to a separate process allowing an independence from strict language and tool-chain boundaries. The communication between components of a node is based on Extensible Markup Language (XML). Using XML allows standards-based interoperability for the components. With this interoperability a module can be started on a given node and its services made readily available to other DTN processes already running on the common core.

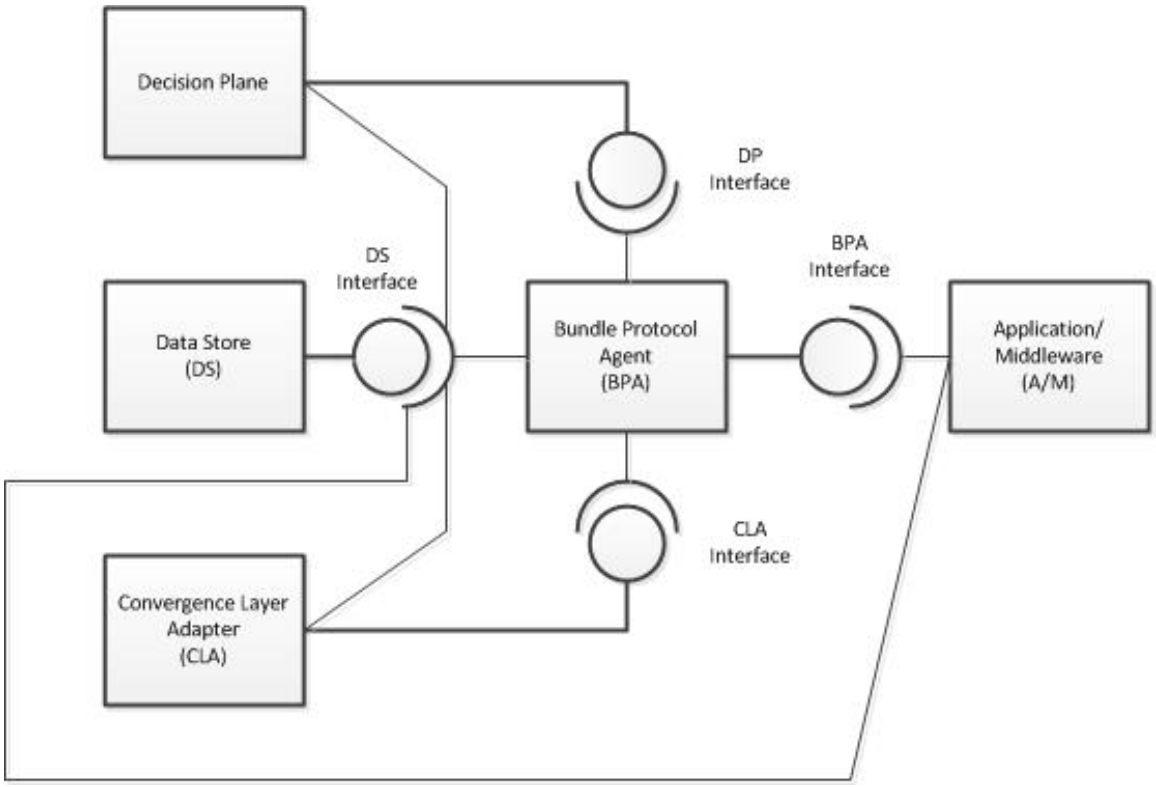

Figure 1. Architectural Components and Interfaces (Redrawn after [6])

1. Bundle Protocol Agent (BPA)

The Bundle Protocol Agent offers the bundle protocol services to the operational node. BPA functions include the primary services such as: bundle forwarding, fragmentation and reassembly, custody transfer mechanisms, delivery to application, deletion, sending administrative bundles such as status reports, and security functions [5]. The SPINDLE BPA is made up of four modules: policy module, router module, naming and late binding module, and content-based access module.

2. Decision Plane (DP)

The decision plane interacts with the BPA and is responsible for key decisions involved in the BPA mechanisms. The DP utilizes the BPA policy module to provide interpretation and enforcement of user-specified policies in the basic functionality. The SPINDLE software uses an "event-condition-action" design. It is intended to mediate the communications between the BPA and the rest of the DP. The router module contains the services provided to the DP to include: unicast and multicast route computation generation of next hop(s) for bundles, replication and forwarding, bundle scheduling, and decision to take custody of a bundle, or to discard a bundle. The router module determines what network state to distribute, and to whom, and when. It gathers the network state from incoming dissemination bundles as well as the local CLAs. The third module in the DP is the naming and late binding module. The naming and late binding module maintains and opportunistically shares naming policies among the DTN nodes in the network. This module resolves names to endpoint identifiers of nodes that are used by the router module. It is also responsible for registration and dissemination or synchronization of name knowledge bases for the DTN system. The last module contained by the DP is the content-based access module. This module is responsible for: content caching/replication, distributed indexing, and content-addressable search. It is capable of using several services from the other DP modules to complete tasks [6].

3. Convergence Layer Adapter (CLA)

Convergence Layer Adapter sends and receives bundles for the BPA. It converts the data transmission service utilized in the local node to the present bundle transmission service determined by the BPA. This adapter allows the local network to integrate multiple topologies and still utilize the SPINDLE software for disruption-tolerant networking. The CLA discovers and retains the information for links encountered while tracking the routing groups discovered.

4. Data Store (DS)

The Data Store module is a persistent storage service for the SPINDLE system. The main job of the DS is storing bundles, knowledge about bundle metadata, network state information and application state information. It is accessible from all other components of the system. There are three types: key-value, database and knowledge base key-value. Key-value type is a simple key-value store that allows other modules to add, retrieve and delete key-value pairs. The database type consists of databases based on a relational database management system that allows other modules to access the information in the DS. It provides the services to add, delete, and get elements with multiple fields that can be searched by key. The Knowledge Base (KB) type of DS uses a KB capable of supporting deduction or inference through execution of rules on stored facts. A KB can support internal data storage or back-end storage system such as MySQL or Berkeley DB [6].

5. Application/Middleware (A/M)

The application and middleware agents simply use the BP services to transmit and receive bundle payloads. Some examples of the applications SPINDLE uses are ***dtnsend*** and ***dtnsource*** which prepare and send bundles to the destination, while ***dtnrecv*** and ***dtnsink*** are used to receive bundles and assemble the packet data unit for the applications. SPINDLE also provides some native administrative applications as well. The three basic applications provided by SPINDLE through the A/M are ***dtnping***, ***dtntraceroute***, and ***dtnreporter***.

D. BBN SPINDLE BPA ALGORITHMS

SPINDLE utilizes two routing algorithms to transfer traffic from one node to the next. The first algorithm is Prioritized Epidemic (PREP) and the second is Anxiety-Prone Link State (APLS). PREP imposes a partial ordering on bundles for transmission and deletion in an individual intermediately node. APLS builds upon PREP by introducing a shortest cost path (SCP) mode triggered when the configured threshold is more than the computed cost to the destination.

1. Prioritized Epidemic Routing

In Prioritized Epidemic, a drop priority of bundles is calculated for bundles that have a hop-count value greater than or equal to a pre-configured threshold. A lower priority is given to the larger shortest-path cost to the destination. For example the drop priority pd(B) of a bundle B is derived from the cost of the lowest-cost path from the current node to the bundle’s destination D. Thus a

higher cost means a higher likelihood for the bundle to be dropped in case of buffer overflow. In addition, a transmission priority is given to bundles that are headed toward their destination. The transmission priority pt(B) of a bundle B is equal to its ranking in a radix sort on (expiryTime(B)-currentTime) and (creationTime(B)-currentTime). All ties are broken randomly and internode costs are based on average availability (AA). The AA metric attempts to calculate and to measure the average fraction of time in which a link will be available for use which is then disseminated to all nodes. Path costs are calculated using the topology learned through this dissemination, with cost of a link l set to (1 – AA(l)) + c (a small constant factor that makes routing favor fewer number of hops when all links have AA of 1) [6].

2. Anxiety-Prone Link State

Anxiety-Prone Link State introduces the shortest cost path mode that is used anytime the cost to the destination is less than a pre-configured threshold. In regions prone to loss of connectivity, or at times when connection is lost intermittently, PREP-like replicated forwarding is employed. Any other times or in stable regions, the behavior of the nodes resembles conventional MANET forwarding. APLS is expected to be highly adaptive and competitive with protocols designed for both disrupted and stable environments [6].

3. Late Binding

Late binding is the process the BPA uses for deferred name resolution. Due to the nature of a changing network

the source's resolution of names cannot be counted on for DTN. SPINDLE uses intermediate nodes in the network (or even the destination) to perform the resolution of names. The late binding module addresses the problem of progressively resolving the destination name to the canonical name. Some of the key components to the SPINDLE late binding module are an expressive name scheme based on a declarative logic language, the addition of a metadata extension block to the bundle protocol to carry information for name resolution, the use of knowledge bases to store name management and resolution information. SPINDLE also uses publish-subscribe mechanisms to exchange name management information and performs the name resolution procedures on each DTN node [6].

4. Disruption-Tolerant Access to Content

SPINDLE proposes that users simply describe what they want, not where it is stored, and the network moves information when and where it is needed. We cannot have this in a standard network due to use of the universal resource locator (URL). In their current form, URLs include a DNS name that must be resolved at the source and utilize technology that expects a reliable connection to the Internet. In the situation of a deployed ad-hoc network, a reliable connection is not guaranteed. To achieve the ideal disruption-tolerant network we need to be able to access information based on content. DTN research is working on approaches for opportunistic caching, indexing, and retrieval of data in such a way that the data can be accessed even though there is no connection to the Internet.

E. BASIC REQUIREMENTS FOR PROTOCOLS

Data traffic moves from one node to the next via commonly understood communications. Data traffic is sent and received on communicating systems to establish communications. Therefore, the protocols should specify rules governing the transmission. In SPINDLE, several common requirements are addressed differently than in TCP/IP.

1. Packet Data Unit Formats for Data Exchange

Data is divided into bit-strings with a ***header area*** and a ***data area***. The TCP header as seen in Figure 2 consists of the source port (16 bits), destination port (16 bits), sequence number (32 bits), data offset (4 bits), reserved field of 3 bits, flags (9 bits), window size (16 bits), checksum (16 bits), urgent pointer (16 bits), options field (Variable 0–320 bits, divisible by 32) and padding of zeros to ensure that the TCP header ends and data begins on a 32 bit boundary [7]. Bit-strings longer than the maximum transmission unit (MTU) are divided in pieces of appropriate size.

SPINDLE uses the Bundle Protocol which defines a series of contiguous data blocks as a bundle. Each bundle starts with a block that has a header to identify the received bundle and all the blocks in it. It also identifies if it is fragmented. The primary block header consists of the version block (8 bits), Bundle Processing Control Flags, Block Length, Destination Scheme Offset, Destination SSP Offset, Source Scheme Offset, Source SSP Offset, Report-to Scheme Offset, Report-to SSP Offset,

Custodian Scheme Offset, Custodian SSP Offset, Creation Timestamp time, Creation Timestamp Sequence Number, Lifetime, Dictionary Length, Dictionary byte array, Fragment Offset, Total Application Data Unit Length, as seen in Figure 3. The Bundle Payload Header is much simpler as seen in Figure 4. All fields in the bundle protocol header other than the version use SDNV which allows them to grow or shrink in length depending on need [8].

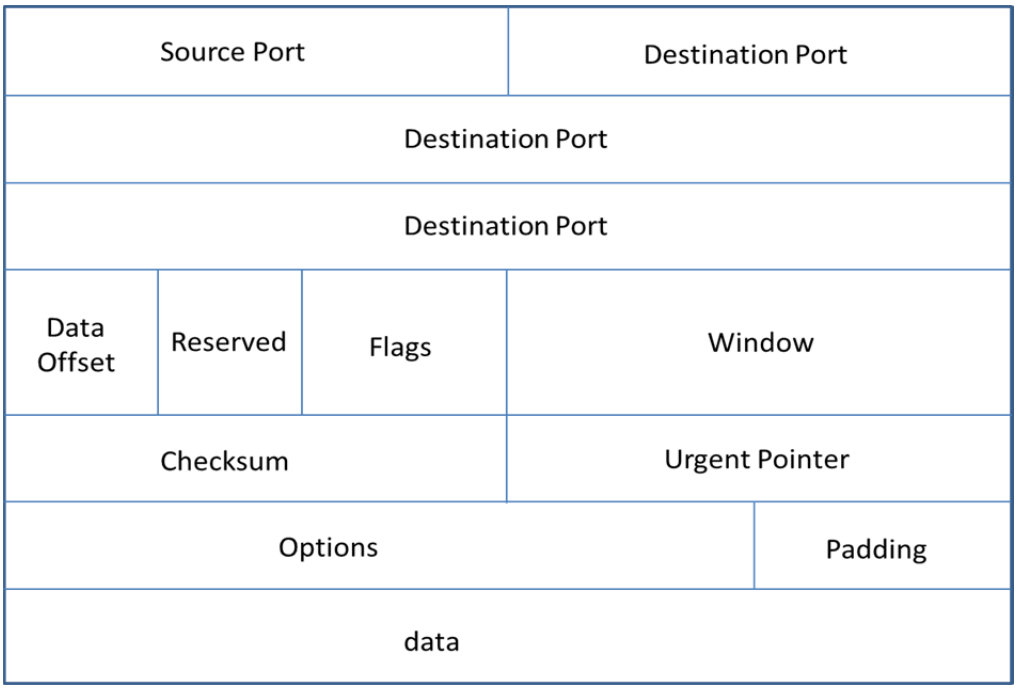

Figure 2. TCP Header Block

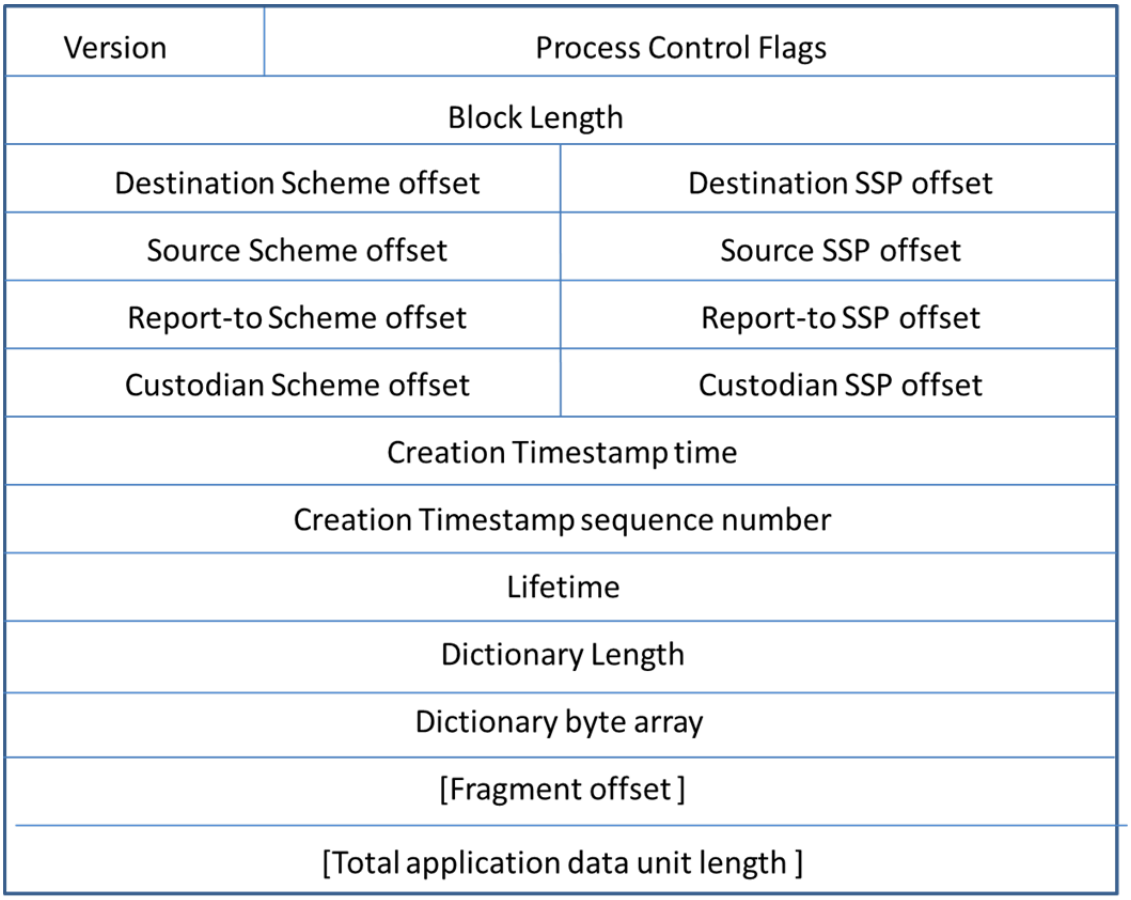

Figure 3. Primary Bundle Header (Redrawn after [8])

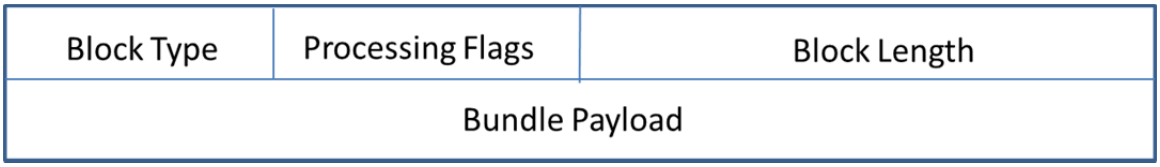

Figure 4. Bundle Payload Header (Redrawn after [8])

TCP accepts data from a data stream, segments it into chunks, and adds a TCP header creating a TCP segment. The TCP segment is then encapsulated into an Internet Protocol datagram. However, this assumes there is a persistent connection to be trusted to remain in effect for the complete duration of the transmission. This is not the case for DTNs. In DTNs, there may never be an end-to-end path

available. In DTNs, disconnection is often the norm rather than the exception. Since this is the case controlled replication becomes very important.

SPINDLE uses the Bundle Protocol to determine the sizes of “bundles” to transport between nodes. Bundles are concatenated sequences of at least two block structures. The first block is the primary bundle block and contains the primary bundle block header. There can only be one of these in a bundle. Multiple additional blocks may follow the primary with the last being the payload block [8].

2. Address Formats for Data Exchange

Addresses are used to identify both the sender and the intended receiver(s). A connection between a sender and a receiver can be identified using an address pair (sender address, receiver address). Addresses are resolved through name resolution databases that can be consulted reliably. Since destination identifiers may not be available at the source and the nodes with such information are not available at the source SPINDLE uses EIDs as addresses for identifying the receiver. Intermediate nodes in the network will perform the resolution of EIDs, referred to as late binding.

A bundle uses three EIDs: source, destination, and reply-to. When the SPINDLE BPA sends a bundle, it must set the bundle’s source EID to the source from which the bundle originated, the destination to which the bundle is being sent, or the address to which bundle status reports concerning this bundle are sent (reply-to) [9].

3. Address Mapping and Routing

Standard network protocols use address mapping to translate a logical IP address specified by the application to an Ethernet hardware address. Internetworking is used when systems are not directly connected to forward messages on behalf of the sender.

The SPINDLE BPA uses a so-called “late binding architecture” which includes an expressive name scheme, an addition to the bundle protocol for name resolution, use of knowledge bases to store name management and resolution information, publish-subscribe mechanisms to exchange name management information, and name resolution procedures that are performed at individual DTN nodes [6].

For routing SPINDLE uses the PREP and APLS algorithms, explained in Chapter II, to determine the best average fraction of time in which the link will be available for use to the SPIDLE BPA software. Testing and comparison to epidemic routing, probabilistic forwarding and purging, and future contact prediction approaches can be seen in Krishnan’s original research paper [6]. Bundles are routed using store and forward over varied network transport technologies (including both IP and non-IP based transports). The transport layers carrying the bundles across their local networks are called bundle convergence layers. The bundle architecture therefore operates as an overlay network, providing a new naming architecture based on Endpoint Identifiers (EIDs) [6].

In many non-IP-based networks, for example X.25, Frame Relay and ATM, the connection oriented communication is

implemented at network layer or data link layer rather than the transport layer. In X.25, in telephone network modems and in wireless communication systems, reliable node-to-node communication is implemented at lower protocol layers.

4. Sequence Control

In traditional networks, long bit-strings are divided into pieces, and sent individually. This results in pieces possibly arriving out of sequence. Sequence information is provided by the sender so the receiver can determine what was lost or duplicated, ask for necessary retransmissions and reassemble the original message.

In the SPINDLE bundle protocol, the primary bundle header maintains the fragment offset indicating the offset from the beginning of the original application data unit from which the payload of the individual bundle was located. So the SPINDLE BPA retrieves each fragment and assembles them in order based on fragment offset.

III. TEST TOPOLOGIES AND METHODOLOGY

A. DISRUPTION-TOLERANT NETWORK TOPOLOGIES

The SPINDLE BPA software developed by Raytheon BBN Technologies provides the BPA support specified in the Bundle Protocol Agent Functional Requirements (RFC 5050) and the Bundle Security Protocol (BSP), as specified in the Internet Engineering Task Force (IETF) Delay-Tolerant Networking Research Group (DTNRG) Internet Draft revision 08 documents [10].

SPINDLE BPA supports five types of topology: linear, ring, edge, grid, and mesh. The linear topology (Figure 5) Connects nodes in a linear formation based on the nodes list specified in the configuration file. Node 1 connected directly to Node 2. Node 2 connected directly to Node 3. Node 3 connected directly to Node 4.

Figure 5. Linear Topology (Redrawn after [10])

The ring topology (Figure 6) is like linear topology, but the ends are also connected. So Node 5 is also connected to Node 1 creating a standard ring network topology.

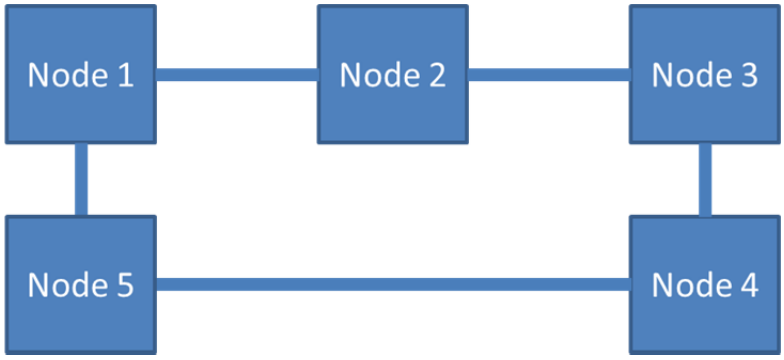

Figure 6. Ring Topology (Redrawn after [10])

The edge topology (Figure 7) consists of a connection between the first node on the nodes list to all other nodes. Thus Node 1 is connected to Node 2, Node 3, and Node 4 but Node 2 is not connected to Node 3 or Node 4. The Node 3 cannot connect directly to Node 4 either.

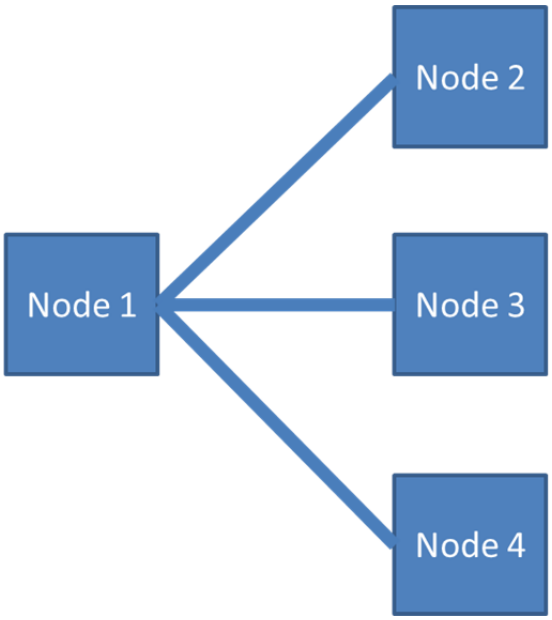

Figure 7. Edge Topology (Redrawn after [10])

The grid topology consists of a NxN grid formation where P = "total nodes" and N = ceil(sqrt(P)). The grid is filled left to right and then top to bottom as seen in Figure 8.

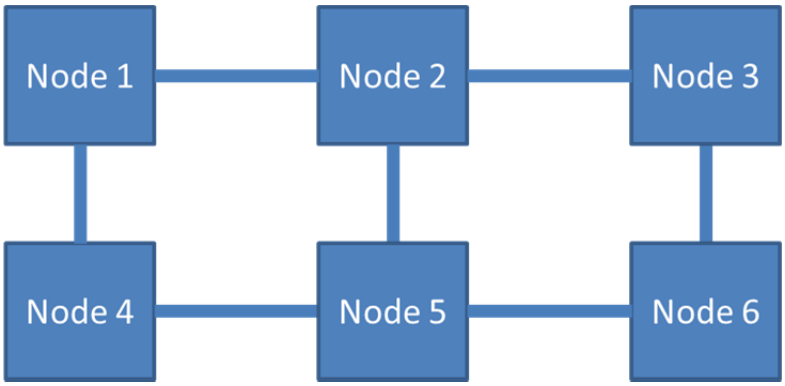

Figure 8. Grid Topology (Redrawn after [10])

In mesh topology, each node is connected to all the other nodes, as seen in Figure 9.

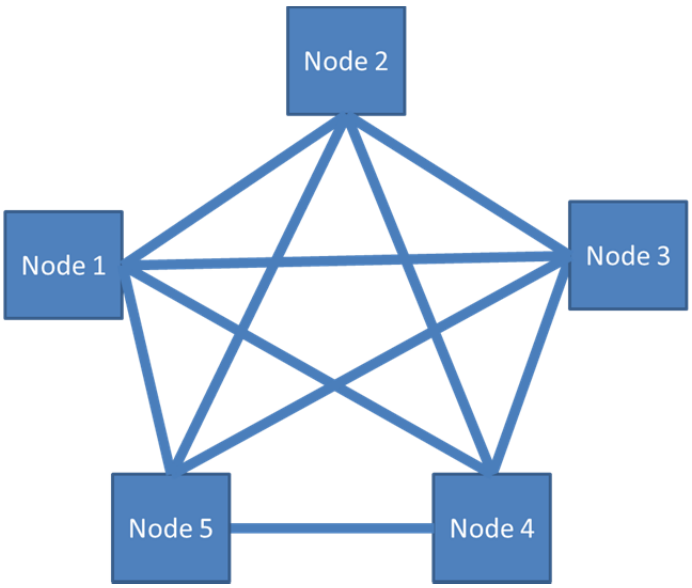

Figure 9. Mesh Topology (Redrawn after [10])

The specific type of topology used for a scenario can be defined in the BPA configuration and changed when needed. The two BPA configuration files, ***bbn-bpa.ips*** (Appendix A) and ***bbn-bpa.config*** (Appendix B), allow the user to configure these various routing scenarios and node topologies quickly. Node topologies can be defined via static routes in the "***topology***" section of the ***bbn-bpa.ips*** configuration file.

By default SPINDLE topologies are set up using static multi-hop routes that are installed based on the topology chosen. Routing can be changed however to non-static routing paths handled by the routers themselves. This option allows mixed-mode setups where a portion of the network uses static routes and another portion uses ad-hoc routers. The other option for routing is to tell the parser to not install any sort of multi-hop routes. Another routing option available is the Neighbor Discovery Options. In addition to the default behavior, Neighbor Discovery can be configured to handle some special cases. The first of these special cases is to tell the parser to install unicast beacons one-way links (i.e., from node1 to node2). A second case is to cause a permanent link to be created at startup between two known nodes. The third case is the unidirectional setting. This setting tells the parser all specified links should be created as unidirectional. Routes can be defined across nodes in this case [10].

B. METHODOLOGY AND TESTING

BBN SPINDLE is new software developed for the U.S. Marine Corps so it has been provided to Naval Postgraduate School (NPS) in Monterey, CA for testing before it is approved for use in a real network environment.

> The SPINDLE BPA software delivers BPA support as specified in the Bundle Protocol Agent Functional Requirements (RFC 5050) and the Bundle Security Protocol (BSP), as specified in the Internet Engineering Task Force (IETF) Delay-Tolerant Networking Research Group (DTNRG) Internet Draft revision 08 documents. [10]

The first release given to NPS is the bbn-bpa-s3-EndemicIRAD-release-20101221-1046. It is the Source Code Release for DTN IRAD on 21 December 2010. As part of the research team our goal was to run the application software on nodes that would simulate an actual operation of a network. To accomplish this we set up three individual router nodes running a Linux-variant Vyatta Core version 6.3 as the operating system and a laptop node loaded with Linux-Mint 12 as the operating system. The SPINDLE BPA package is a 32-Bit package that was originally tested on Ubuntu 8.04 and Ubuntu 8.10. Since the original release it has also been run by other testing teams on Ubuntu 10.10 and 12.04. According to the user manual there are no specific pre-requisites for the application layer to run the SPINDLE BPA software and the SPINDLE DTN applications packages [10].

The routers were DTN nodes connected by multiple parallel links through separate hubs to simulate fail-over from one port to another and finally test the disruption-

tolerant networking BPA. Remote access to the testbed through secure shell (SSH) was used so that the tests could be observed and repeated by remote evaluators.

The second portion of my work was to analyze the source code for the Ekiga softphone software to assist in future testing of the BBN SPINDLE BPA software. The goal was to write code that sent and received ICMP messages that are altered to accommodate disruption-tolerant network traffic messages. This will consist of using an ICMP type value currently unused in common software. The ICMP type message will have multiple code values allowing the receiving code to discern the estimated duration of the disruption to the network and send a message to the user. Ultimately the message will be routed through the chat agent. For now the message will be displayed in the command line window.

IV. BBN SPINDLE SOFTWARE TESTING RESULTS

After initial testing began and a testbed was setup for use with the SPINDLE BPA, BBN provided the "Software User Manual for the USMC Disruption Tolerant Networking" with a publish date of 14 September 2012. Several of the tests described here can be found in greater detail in that user manual. [10]

A. INSTALLATION AND CONFIGURATION

The BBN SPINDLE BPA software is provided as a single compressed package for the user. You begin by creating a folder to easily identify the DTN software on the system. Copy the bbn-s3-20101221.tgz file provided by BBN to the file on the system where you plan to run the software and unpack the file using the command:

```
$ tar xvzf bbn-s3-20101221.tgz
```

Next step is to extract the SPINDLE BPA and SPINDLE DTN Applications software in the current directory. Once bbn-s3-20101221.tgz is unpacked you will have a folder with two files in the folder: bbn-bpa-s3-EndemicIRAD-release-20101221-1046.tgz and bbn-dtn3exp-20101216-1606-r5140.tgz. Open the new folder and unpack the two files that the bbn-s3-20101221.tgz uncompressed.

The file bbn-bpa-s3-EndemicIRAD-release-20101221-1046.tgz unpacks the SPINDLE BPA software module including the configuration files and topology files. From this folder you will start and stop the BPA module. This folder contains all the files required to run the BPA module. In the folder unpacked from the bbn-dtn3exp-20101216-1606-

r5140.tgz file, we install the SPINDLE DTN Applications software. The SPINDLE DTN Applications software consists of the following applications: ***dtnapitest, dtncp, dtnping, dtnrecv, dtnsend, dtnsource, dtncat, dtncpd, dtnpoll, dtnreporter, dtnsink, dtntraceroute***.

To be able to use the packages unpacked and installed the bbn-dtn3exp-20101216-1606-r5140/bin directory will need to be added $PATH environment variable. Once this last step is completed the native applications provided by the SPINDLE DTN can be run from the command line. The most complicated part of the install is adding the bbn-dtn3exp-20101216-1606-r5140/bin directory to the $PATH environment variables. This requires the user to be familiar with the operating system and have a working knowledge of how to change the user profile.

The BBN SPINDLE BPA package includes two configuration files: ***bbn-bpa.config*** and ***bbn-bpa.ips***. The ***bbn-bpa.config*** file allows for setting Convergence Layer Adapters (CLAs), Neighbor Discovery (ND) adapters, and Routing components. The ***bbn-bpa.config*** file has several additional configuration options that can be manipulated by the user such as:

- Location of bundle storage and bundle state files
- Location of BPA and routing logs
- Location of binaries for BPA, Routing processes and CLAs.
- Set the log level for each component in the SPINDLE BPA package.

The ***bbn-bpa.ips*** is initialized with local node address definition and it allows the user to define the topology of the network to be used. Since it is used to define a specific topology to be used by the SPINDLE BPA software it supports multiple configuration parameters defined by the user to better clarify the topology operations. The default topology of the SPINDLE BPA is to attempt neighbor discovery. This file also allows the user to define interface groups for both multicast one-hop beacons and unicast beacons.

The BBN SPINDLE BPA package supports the following routing topologies:

- Static Routing: Static routing definitions are used with no discovery of network information, which allows for both connected and disconnected routing.
- Epidemic Routing: Disconnected Routing is dynamic and uses epidemic dissemination to distribute bundles.
- Link State Routing: Connected Routing topology is dynamic and accomplished by using the TMRG router to determine routing.
- Endemic Routing: Combines the Connected Routing and Disconnected Routing ideally for DTN networks.

Endemic Routing is operational in the 20101221-1046 release. SPINDLE BPA uses a hybrid routing mechanism that combines Connected Routing (CR) and Disconnected Routing (DR) to allow transition between each mechanism, any number of times, to accommodate for the state of disruption in the network. In theory, with DR a network can support traffic

can be reliably routed through parts of a network even though they are disrupted. With CR techniques a network can route bundles efficiently through connected parts of the network. CR techniques include a combination of static and dynamic routing. Endemic Routing's configuration can also be customized by modifying the configuration parameters in the ***bbn-bpa.config*** file.

B. STATIC ROUTING TOPOLOGY

Static Routing topologies use static routing definitions only and do not discover any network information, other than what is configured. Static routing allows for both connected and disconnected routing scenario because the routing information is hard-coded via the configuration.

At this stage of testing our testbed was setup with only the four nodes so I began by running the first two scenarios in the user manual provided by BBN to compare the actual operation with the advertised operation. Scenario 1 utilized a three node linear topology with static routing. This scenario described the setup, and running of a three node linear topology using static routing. In Scenario 1, bundles were exchanged between the end-point nodes demonstrated in Figure 10.

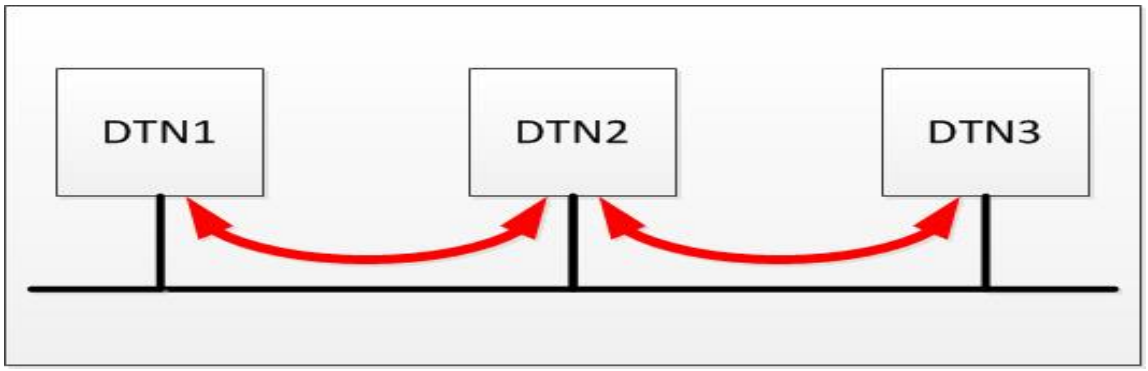

Figure 10. Static Routing Topology (Redrawn after [10])

To configure the static routing on the nodes the user utilized the two BPA configuration files and manipulated the variables inside them. The first variables I set up were the node identification addresses. For static routing each node knows of all the others in the routing topology at startup giving it knowledge of all the routing available. Since all the nodes are configured to be aware of all the other EIDs in the network the routing can be direct and optimized. If a node is identified in the network but found to be unreachable each node can recalculate a path by referring to the network topology and node EIDs.

For this configuration I tested bundle traffic through the ***dtnsend***, ***dtnrecv***, and ***dtnsource*** commands in the SPINDLE BPA. The first test was to send a simple message from the originator to another EID in the network. The receiver node was set to receive bundles directed toward the EID identified in the ***dtnrecv*** command in the form of: ***dtnrecv [opts] <endpoint>***. The ***dtnrecv*** command tells the agent to enter an endless loop waiting for traffic to arrive. Bundles were sent from the source node though the ***dtnsend***

command which is in the form of: ***dtnsend [opts] –s <source_eid> -d <dest_eid> -t <type> -p <payload>***. As the first test the message is generated by the command line options at the originating node as the payload. It was simple and consisted of two words, “Test message.” Ideally it would travel through one node before reaching the endpoint. The second test was to send a text file from the originator to the endpoint. This text file contained two sentences to demonstrate a file with multiple characters in it.

Since both the test message and the text files were small neither needed more than one bundle to convey the data to the receiver endpoint. The third test was to send multiple bundles from the originating node using the ***dtnsource*** command which requests custody transfer for the bundles along the path to the destination and is in the form of: ***dtnsource [opts] -s <source_eid> -d <dest_eid> -b <bundle size> -n <num bundles>***. For the test I set the number of bundles to be 5 and the bundle size to 1024 bytes. At the destination I observed in the log file all the information about the bundles received. The last test was to demonstrate how the SPINDLE BPA sends a large bundle that will fragment and the use of custody to ensure that all fragmented bundles are delivered. The SPINDLE BPA fragments bundles to fit into the MTU of the link before sending, so we can see the fragmentation being applied in the log files.

At the destination I reviewed the log file to verify the performance of the SPINDLE BPA and compare it to what the user manual states. For the static topology all the

tests ran the same as documented in the published user manual. I saw no difference between my results and what the user manual stated should be in the log files. This confirmed that while in a static topology the SPINLDE BPA software operated as advertised and no problems were noticed for sending and receiving traffic.

C. EPIDEMIC ROUTING TOPOLOGY

Epidemic Routing is a Disconnected Routing topology. This topology is dynamic and is accomplished by using the DR router, which discovers routing and uses epidemic dissemination to distribute bundles. DR nodes coordinate with neighbors in distributing bundles to all nodes in the network, resulting in epidemic bundle distribution.

In Scenario 2, three nodes are set up for a Dynamic Routing topology as seen in Figure 11. The three nodes are configured similar to the static topology with the exception of the individual nodes being aware of the network EIDs adjacent to them at startup. For the dynamic setup the configuration file ***bbn-bpa.ips*** of a router is changed from listing each node in the DTN network to only being aware of itself. By leaving the topology configuration blank in the SPINDLE BPA ips configuration file, the router will determine the topology based on the systems interfaces and routing algorithms. The dynamic routing is identified in the config configuration file when you set the routers variable to "***dr.***"

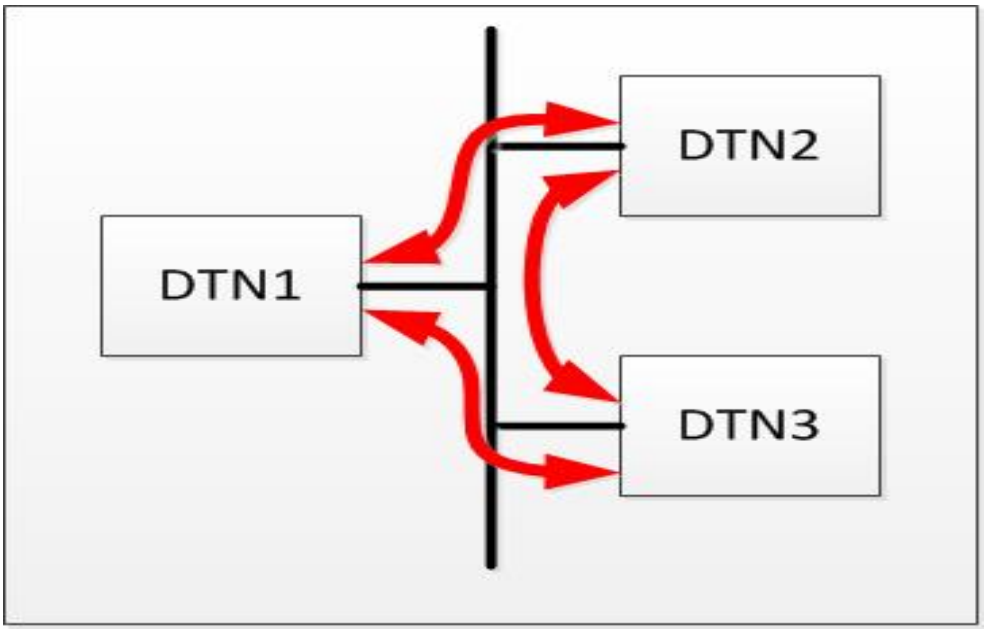

Figure 11. Three-node Dynamic Routing Topology (Redrawn after [10])

In the SPINDLE BPA, the epidemic routing algorithms are used for the dynamic routing. A simple explanation of epidemic routing is to imagine a situation where a person has started a rumor by telling a select few of the people around them. Assume everyone is in a small conference room and this fixed population is of size n. For simplicity of explanation, assume homogeneous spreading. The simple epidemic theory is that anyone can tell anyone else with equal probability. Assume that k members are already aware of the rumor and that the rumor spreading occurs in rounds. What is the probability of hearing the rumor is $P_{rumor}(k,n)$, assuming that a particular unknowledgeable member is told in a round if k are already in a round and if k are already infected? It is $P_{rumor}(k,n) = 1 - P(\text{nobody tells any others})$ which equals $1 - (1 - 1/n)^k$. The expected number of newly

told members is equal to (n-k) * $P_{rumor}(k,n)$. In short, it demonstrates the Binomial Distribution.

With the dynamic routing configuration setup I began the tests on traffic. I started the SPINDLE BPA on all nodes enabling the bundling protocol. For these tests the receiver node used the ***dtnsink*** command which tells the agent the exact number of bundles to expect from the traffic and is in the form of: ***dtnsink [-n num|-x|-u|-c] [opts] [EID] … [EID].*** I then logged into the sending node and for this test I first sent five bundles of 1024 bytes to the remote destination using the ***dtnsource*** command.

Next, I executed a new set of commands to copy a file with bundles using the ***dtncp*** and ***dtncpd*** applications. These applications are provided with the SPINDLE BPA and are the SPINDLE BPA equivalent to the commonly used command line protocol secure copy command (***scp***). The ***dtncpd*** and ***dtncp*** applications are used to send a file via DTN to a pre-defined path on the target node. At the receiver node I used the ***dtncpd*** application command in the form of: ***dtncpd [directory].*** This command directs the receiver to collect incoming files and places a copy in a specified output directory. At the sender node I used the ***dtncp*** application to send the designated file. The ***dtncp*** command form is: ***dtncp [-expiration sec] [-D] [-C] [-p #] <filename> <destination_eid> [remote-name].***

I reviewed the log files of both the sending nodes and the receiver nodes to verify the performance of the SPINDLE

BPA and compare it to what the user manual states. In this lab environment, all the tests ran as expected for the send and receive.

D. ADDITIONAL TESTS CONDUCTED

Since these scenarios did not test more than the simple transmissions of one node to another we needed to test other functionality of the BBN SPINDLE BPA. In this other testing, we found issues that were reported to BBN as bugs in the software.

Our initial testing found compilation issues due to unused variables with current compilers. Using recently updated Fedora, Ubuntu, and Debian systems in our testbed we attempted to compile the software and in each case compilation errors are reported. We were able to get it to successfully compile by removing the unused variables but we surmise that some of them are required to check a return value for error conditions and various other operations. The new release of BBN SPINDLE BPA compiles cleanly with no errors or warnings output. It includes the source files for both 64 bit and 32 bit operating systems. This was tested on an Ubuntu 12.04 LTS operating system. After the software was installed with no errors and the environment variables set for the commands I was able to operate the applications from the command line.

In release 20101221-1046, when the BBN SPINDLE BPA software was unzipped and installed on a client machine operating a newly installed operating system, it would not operate. Four separate versions of Linux-Mint operating systems were installed and the BPA was installed on each

one to test the validity of the error. After I tested the fourth version of Linux-Mint operating systems and found the error persistent it was concluded that there was a vital part of the install missing. After we informed BBN of this error they found it was a library missing from the newly install operating systems. The SPINDLE BPA would not start up due to vital dependencies missing. BBN found that the ***libpcla.so*** binary was missing from basic builds of the software. Since this is not included in most Linux based operating systems as an initial library the SPINDLE BPA did not operate after a successful install. The LIBPCLA build has been augmented and the ***libpcla.so*** binary is now included in every build of release 20130109.

Release 20101221 had an issue with listening and transmitting on some nodes when dynamic routing was involved. In the SPINDLE BPA, when DR was enabled, the transmission from the first application would not be sent over the wire by DR and only delivered locally. This issue was caused when a DTN application was sending to the multicast group while another local DTN application was listening to the multicast group. Release 20130109 can provide this functionality with no issues now. When tested the software accepted transmissions seamlessly. Tests were conducted on release 20130109 with three nodes running the BPA software. The first test had two of the three nodes enrolled in the multicast group while the third was not enrolled in any multicast groups. All three nodes were given the command to listen for bundles. A multicast bundle was broadcast from the node not enrolled in the group to the members of the multicast group. All the enrolled

members of the group received the bundle correctly. Next step was to disconnect the node providing the connection from the sending node to the group and attempt sending again. The bundle was held back and did not get broadcast until the connection was re-established. For the next test the sending node was added into the group. A broadcast from this node sent to the same multicast group was received by everyone in the group as well as the sending node. To ensure this was operating correctly the final node in the network opposite the first sending node was used as the sender. All the tests were completed successfully in the new release.

None of the basic scenarios test the admin reports and bundle processing. To test functionality of the applications included in the SPINDLE BPA we tested the custody reports, ***dtnping***, ***dtnreporter***, and ***dtntraceroute***. ***dtnping*** operates in the new release returning the sequence number of the bundle sent as well as the time to return. This is the expected performance for a ping. However, the other applications do not complete operation. ***dtntraceroute*** bundles are deleted from the traffic by the admin function of the host node before departing. ***dtnreporter*** will register and start the receiving loop but a successful ping from another node is not detected or logged as it should be. Custody of bundles is a problem for release 20101221. DR would not accept custody of bundles and send bundles with the custody bit set. Custody transfer and all receipt types failed to work. These errors resulted from not setting a singleton bit on admin bundles and we expected it to be corrected the new release. Release 20130109 lists

this as one of the changes made. The new release of the BBN BPA sets the singleton bit for all admin bundles but between ***dtnping***, ***dtntraceroute***, and ***dtnreporter*** only ***dtnping*** is operating properly. In the new release, DR now accepts custody of bundles and sends bundles with the custody bit set. However, the method used bypasses the custody signaling process. The change report states that this will allow custody bundles to switch from CR to DR seamlessly. To accomplish this custody transfer the startup script will always set the flag to "enabled" as default.

To further our testing we split our DTN into two networks with different subnets. The main network our routers were communicating on used the addresses of 10.0.1.xx while our secondary network used the addresses 10.2.1.xx to communicate. This should not have been an issue since the user manual gives us a method to let the two networks discover each other by distinguishing the two multicast beacon groups on one of the nodes allowing discovery to be passed along the communication paths to either side. However, in action it was found that the bridging node would not discover the second multicast beacon group. All traffic would transfer internally on either network but it would not bridge between the two networks. A static neighbor identification had to be configured on the bridge node at startup to allow discovery to occur and thus to allow traffic to pass from a node at 10.0.1.1 to 10.2.1.2. This was without a disruption or communication error to the network.

THIS PAGE INTENTIONALLY LEFT BLANK

V. ADDING ICMP RECOGNITION OF DTN TO EKIGA

Testing of the BBN DTN software is only the first step in the testing plan in preparation for deployable software. The next step is to test applications using the DTN software. Chat software is the first application type for this test as it is delay tolerant and regularly used in the military environment as a speedy and reliable method of communicating between units.

A. TEST APPLIATION: EKIGA SOFTPHONE

Ekiga is a VoIP, IP Telephony, and Video Conferencing application that allows users to make audio and video calls via Session Initiation Protocol (SIP) or H.323. Ekiga is the first Open Source application to support both H.323 and SIP, as well as audio and video. Some of the main features advertised for the Ekiga Softphone version 4.0 are free Instant Messaging through the Internet, choice of the service provider, SMS to cell phones if the service provider supports it, multi-platform support (Windows and GNU/Linux), and wide interoperability. Ekiga uses the main deployed standards for telephony protocols (SIP and H.323) and has been tested with a wide range of softphones, hardphones, PBX and service providers [11]

Ekiga advertises that it is compatible with any software, device or router supporting SIP or H.323. Ekiga is licensed under the GPL license. As a special exception, permission is given to combine this Ekiga with the programs OPAL, OpenH323 and PWLIB. The combination of software can be used and distributed with manipulations without applying

the requirements of the GNU GPL to the OPAL, OpenH323 and PWLIB programs, as long as the requirements of the GNU GPL are adhered to for all of the software in the combined form. As a future testing tool Ekiga softphone was decided on for the chat traffic testing. The open source code and GNU license allows us to alter the code as desired and test on a DTN.

The chat function is the portion of the software that is significant to my studies. Testing SPINDLE DTN for disruption-tolerant traffic will be done through Ekiga chat first and lead to other traffic applications that can take advantage of the additional DTN transport. My part to prepare the Ekiga softphone for operation on the SPINDLE DTN is to develop an ICMP based method for notifying an application of the use of the DTN transport for its traffic at an intermediate router.

To utilize the Ekiga softphone for DTN chat testing there needs to be a way to receive a message back to the application announcing the disruption. An announcement should be made visually allowing the application to tell the user there has been a problem discovered in the connection. In a standard Internet connection, this could be a termination of the chat and the user would need to re-establish the connection to continue. In a DTN, this would not be needed to continue the chat by virtue of the expectation of delays and disruptions in the connection between stations. I began with the intention to intercede at the socket and intercept ICMP type packets at the socket and add an additional ICMP type to the recognized list. This type would carry the DTN disruption notification and

also the code to identify the expected duration of disruption. The ICMP type value 192 was chosen since it is not already dedicated to a recognized ICMP report. [12] Under the ICMP type value of 192 there were seven specific code values that would be assigned to messages about the connection. Code values would report the messages seen in Table 1.

Type		Code	
Value	Meaning	Value	Description
192	Explicit Network Disruption Notification	0	Explicit Loss Notification
		1	Path re-established
		2	Explicit Delay Notification < 10 s
		3	Explicit Delay Notification < 1 min
		4	Explicit Delay Notification < 10 min
		5	Explicit Delay Notification < 1 hr
		6	Explicit Delay Notification unknown
		7–255	Reserved

Table 1. Example ICMP/ICMPv6 Disruption Messages

As I followed the path of the software and attempted to discover where in the application the ICMP messages were handled I found that Ekiga did not handle the connections internal to the core source code. The connection establishment and utilization is handled by two libraries that are included with the Ekiga Softphone application.

The OPAL VoIP Library is made up of two individual libraries that originally began as separate projects but

have been released in the most up-to-date version as a single library under the title of OPAL VoIP Library.

1. OPAL Library

The first library is the Open Phone Abstraction Library (OPAL). Opal was developed as a derivative of the OpenH323 library. OPAL supports a wide range of commonly used protocols utilized to send voice, video and fax data over IP networks rather than being tied to the H.323 protocol. Initially OPAL only supported the H.323 and SIP protocols but has grown since the initial conception.

OPAL is a C++ class library for normalizing the numerous telephony protocols into a single integrated call model. It is released under the Mozilla Public License. In the OPAL library, the application layer is presented as a unified model for making calls over the underlying protocol or hardware so the calls can be placed and media flow handled (Figure 12).

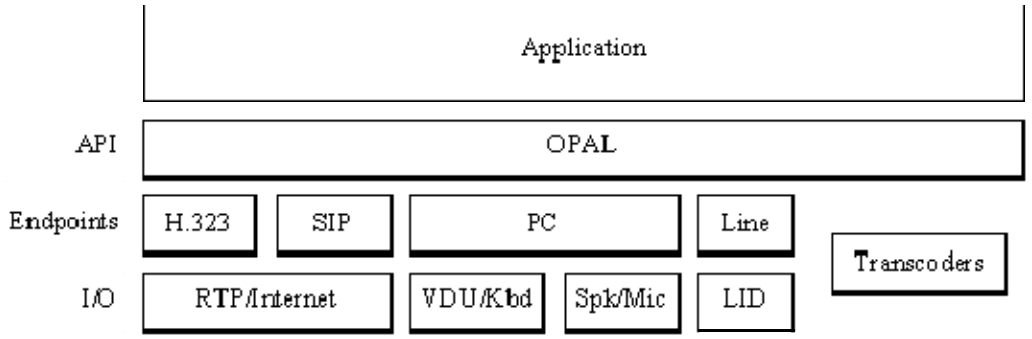

Figure 12. Diagram of OPAL Library (Redrawn after [13])

A call in OPAL is defined as the connection between two or more hosts. Each call is through a connection created by an endpoint. An endpoint is the state

information of the permanent aspects of a telephone abstraction for a particular protocol. This is usually the individual machine that is utilizing the application and the endpoint only lasts as long as the current call. Ekiga can control the connection between endpoints according to its own logic. [14]

OPAL utilizes the ***PTLib*** portable library that allows OPAL to run on a variety of platforms including Unix/Linux/BSD, MacOSX, Windows, Windows mobile and embedded systems.

2. PTLib Library

Portable Tools Library (***PTLib***), which was previously Portable Windows Library, is the second of the two libraries bundled in the OPAL VoIP Library. PTLib is a C++ class library that originated as a method to produce applications that run on both Microsoft Windows and Unix X-Windows systems. PTLib is used for both commercial and Open Source products. The motivation in making PTLib available as Open Source was primarily to support the OpenH323 and OPAL projects. PTLib is separated into two types of classes: Base Classes and Console Components. The Base Classes contain all of the essential support for constructs such as containers, threads and sockets that are dependent on platform specific features. The Base Classes are required for all PTLib programs. [15]

The Console Components employ the library's functionality. This functionality is typically platform independent, so they may not be required for all programs. On Windows platforms the Base Classes and Console

Components can be divided into distinct library archives. In contrast to the Windows division of the libraries, Unix platforms combine all of the code into a single library and rely on the linker to omit code that is not required. [15]

B. HANDLING OF ICMP MESSAGES

The Internet Control Message Protocol (ICMP) is designed to handle errors and exchange control messages. [16] ICMP can be used to determine if a host on the network is responding. ICMP echo request packets are sent to hosts and if the host receives that packet, it will return an ICMP echo reply packet. A common implementation of this is the ***ping*** application, which is included with many operating systems. ICMP is used to communicate status and error information available, including notification of network congestion and of other network transport problems. Ekiga uses ICMP as a tool in diagnosing host or network problems.

ICMP messages can be widely categorized into two kinds of messages, query messages and error messages. For ICMP query messages we can include echo request and echo reply, time stamp request and reply, information request and reply, and address mask request and reply. For ICMP error messages we can include destination unreachable, source quench, redirect, time exceeded, and parameter problems.

A “type” field in the ICMP header identifies different types of ICMP messages. For each “type” field, there may also be a “code” field which acts as a sub-type. For example, echo reply has a type of 0 and code of 0 while echo request has a type of 0 and code of 8. Even though ICMP runs over the IP protocol it has no port numbers.

Unlike TCP or UDP, ICMP messages are sent and received through the so-called raw socket. [17] When an ICMP message is delivered, the receiving host might respond internally but might not communicate back to the source. Since Ekiga like many applications utilizes the kernel to handle ICMP messages it is possible that it does not receive the messages from the raw socket since the kernel may filter unknown ICMP types. Ekiga relies on the operating system kernel to pass it the known ICMP types. This presents a problem when it comes to sending ICMP Messages that are not recognized by the receiving host. It could discard any unverified types and not respond or forward the message up to the application. As an untrusted ICMP Message the host could also deem it as an ICMP attack. To mitigate ICMP attacks most professionals recommended that a host block all ICMP messages of outside of the known error report and query types.

1. Design

Using applications written in C++ code we can test whether this is the case and also test a way around this problem. Since I suspected the kernel of dropping unknown packets and passing known packets the key was to find a way to utilize known packets to carry the information about the connection that is unique to DTN. One technique that may be possible to convey the information and be allowed to pass through the kernel is to use the known type of destination unreachable. ICMP destination unreachable is an ICMP type 3 message consisting of the type field (bits 0–7) set to 3, the code field (bits 8–15), the checksum, an unused field, and lastly the IP header with the first 8 bytes of the

original datagram. [16] Destination unreachable messages are generated by the host or its inbound gateway to inform the client that the destination is unreachable for some reason. There are several reasons a destination unreachable message can be sent. These reasons may include:

- physical connection to the host does not exist
- the indicated protocol or port is not active
- the data must be fragmented but the 'don't fragment' flag is on

To get DTN information through to the application this type of ICMP message seems to have the most potential to convey the message of expected delay. The code field values proposed previously for the ICMP type 192 DTN specific messages from Table 1 can be encoded into the code field of an ICMP destination unreachable message. With unique messages identified by the receiving application a DTN message could be published to the chat window advising the user of a disruption or delay and expected duration until connection is re-established.

I wrote two applications utilizing raw sockets in C++ code. I began by writing an application called "crafter" to send ICMP echo request and receive echo replies. This version crafts an ICMP header and IP header as well as calculates a correct checksum as seen in Appendix C. After that application is complete and tested, I wrote the code to craft ICMP packets of different types and codes in order to verify whether the kernel drops ICMP messages of type 192. The same code can be used to change the ICMP type to 3

which is the type recognized as ICMP Destination Unreachable. I will test to see if the crafted packets pass through the connection and are recognized by an application on the receiving host. This receiving application, called "sniffer," monitors the receiving connection for ICMP messages and outputs to the command line if it recognizes the type and code of each message. The sniffer code is detailed in Appendix D.

2. Experimentation

The first step was to test for what a standard install of Ubuntu 12.04 would do when an ICMP echo request was received. Using Wireshark on two separate hosts running Ubuntu 12.04 (Figure 13), I started captures on both operating systems to observe the traffic across the network Ethernet connection. My first test was to use the native "ping" application to send ping echo requests between the hosts to ensure they are properly connected. As expected the request was acknowledged by the receiving host and an ICMP Reply was sent back. The ping application display showed a fluid request and reply for each packet sent from either host. A sniffer application was on the receiving host tallying up received ICMP packets captured as seen in Figure 14.

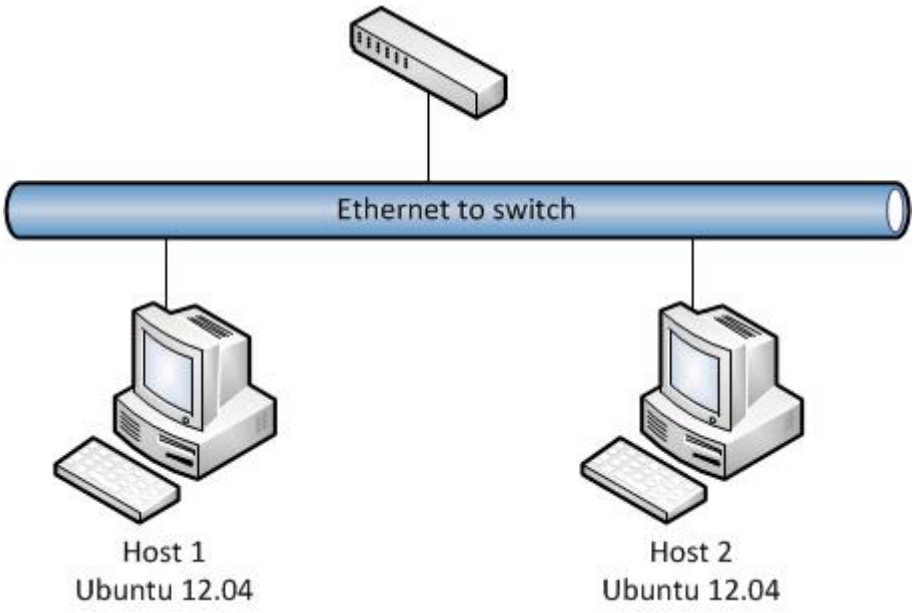

Figure 13. Test-bed setup of host machines

```
Starting...
Waiting for Packets...

TCP : 0   UDP : 0   ICMP : 16   IGMP : 0   Others : 0   Total : 15
```

Figure 14. Displayed results from Sniffer application

The next test was to use C++ code to access the raw socket and send ICMP echo request packets into the connection to simulate a ping request and observe if the same results are captured (Figure 15). The request was acknowledged and an ICMP reply was sent back by the receiving host. This shows that the first version of crafter code runs fluidly with a known type and no extra data to interpret. The Wireshark capture confirmed that the echo request was crafted correctly by the application (Figure 16).

```
Source address: 172.20.109.81
Destination address: 172.20.105.248
Sent 28 byte packet to 172.20.105.248
Received 28 byte reply from 172.20.105.248:
ID: 51047
TTL: 64
```

Figure 15. Command line for crafted ICMP Echo Request

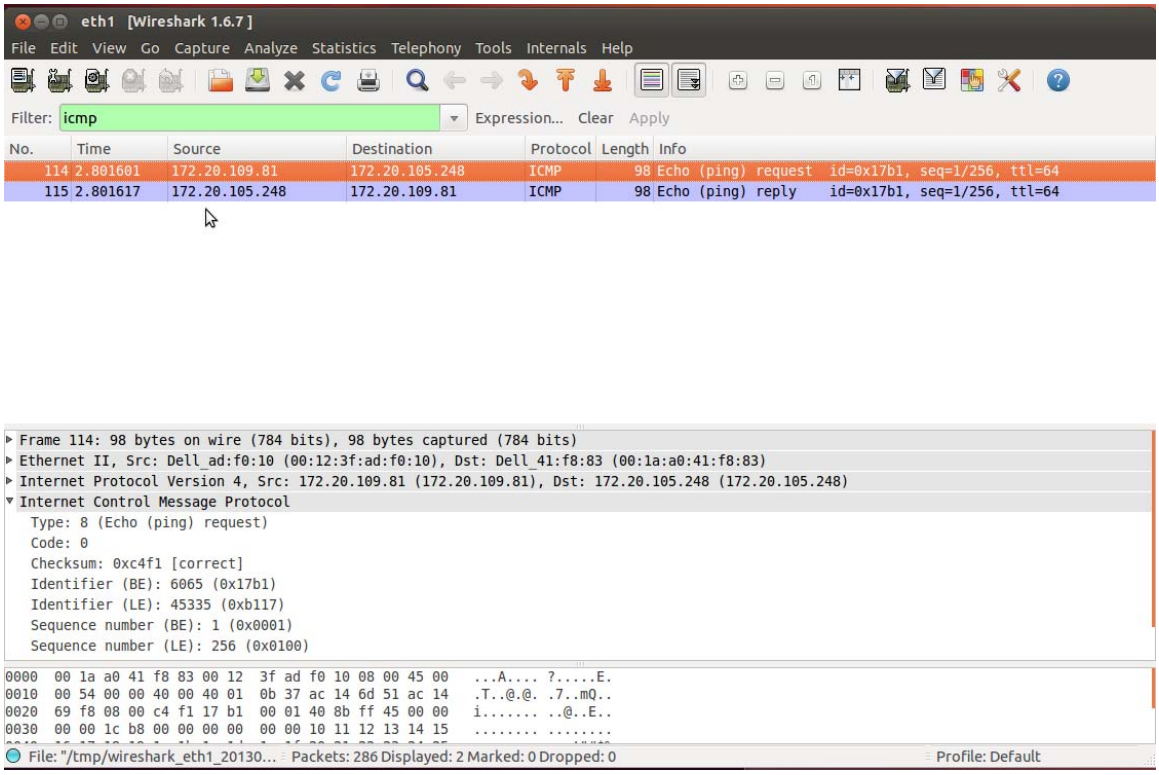

Figure 16. Wireshark capture of crafted ICMP Echo Request at receiving host

The third test was to replace the ICMP type code with our designated ICMP type code of 192 and observe the results. To simulate an ICMP packet of a different type I used a code named pinger found on a programming forum board [17]. Originally this was a ping imitation program. It sends an ICMP ECHO packet to the server of your choice and listens for an ICMP REPLY packet. I used this code as a base and wrote code to let me change the type and code

fields but I reused the ICMP packet forming as well as the correct header and checksum algorithms. Using this code the ICMP header type was replaced with the ICMP type 192 to simulate the new ICMP packets on the line. I inserted the ICMP packet with the ICMP type 192 into the traffic and observed the Wireshark capture on both sending and receiving hosts. Wireshark captured the packet at both ends and reported it as "Unknown ICMP (obsolete or malformed?)" as seen in Figure 17. The crafted packet was sent with no error as seen in Figure 18. This disproves my hypothesis that an ICMP message with an unknown type would be dropped by the kernel. So it could be feasible to use the ICMP type 192 to send DTN specific messages to hosts. Figure 19 shows that the sniffer application can recognize the ICMP type 192 and print a message based on the code value.

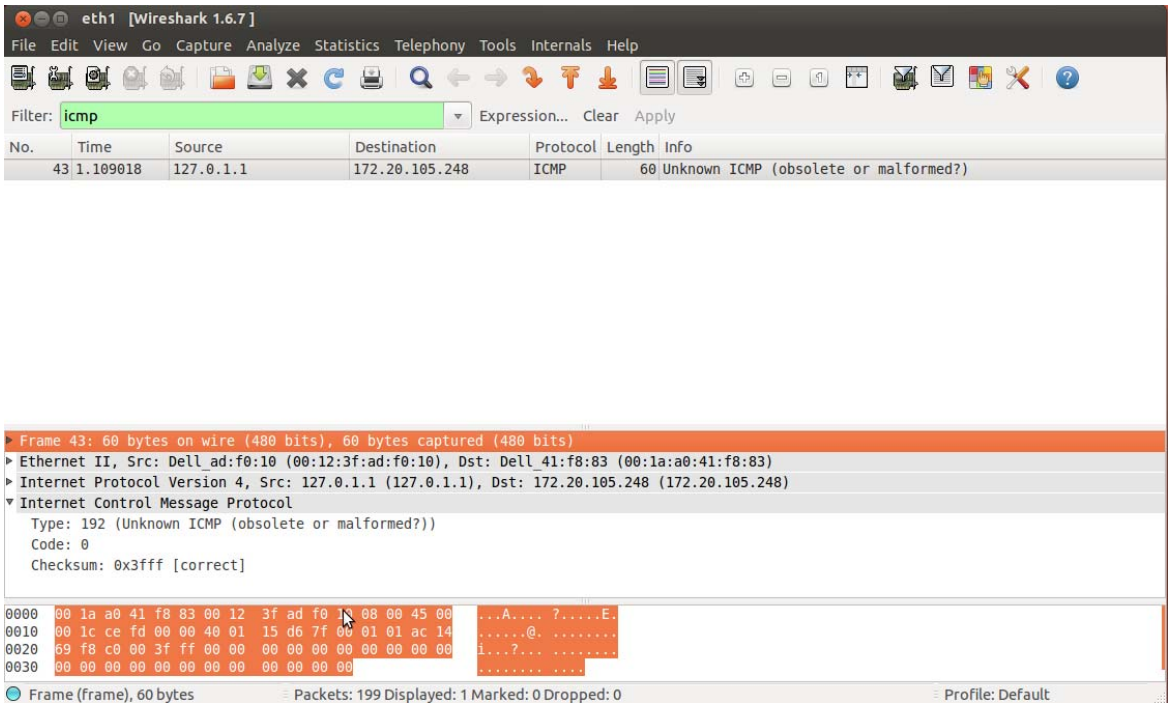

Figure 17. Wireshark capture of ICMP type 192 packet

```
Source address: 172.20.109.81
Destination address: 172.20.105.248
Sent 28 byte packet to 172.20.105.248
Packet type: 192
Packet code: 20
```

Figure 18. Command line output of crafted ICMP 192 packet

```
Starting...
Waiting for Packets...


Received ICMP packet of type 192. Success!
Packet has code value of 0
Explicit Loss Notification
```

Figure 19. Sniffer output from ICMP packet type 192

Once these test were completed I evaluated the efforts it would require to implement this solution at a large scale. Using the ICMP type 192 can be a viable method for conveying DTN specific messages to a host but I asked if there was another method that utilizes current known types that could provide the additional messaging capabilities desired. I looked into other message types to see what would best fit the response required for the DTN unique messages. I settled on ICMP Destination Unreachable as a close relation to disrupted traffic. This second method could be considered here allowing the applications installed on the individual hosts to make changes to already known packets. We can change a portion of an ICMP message already recognized by the kernel. This ICMP message could carry a message that could be parsed out and represent an announcement. This would be similar to a

covert communication channel between the two hosts. The method I propose is to use ICMP destination unreachable type messages and with new "code" values to convey DTN specific messages.

To test this hypothesis I revised the crafter code to generate the required ICMP destination unreachable messages. For the ICMP type I set it equal to 3 designating it to be ICMP destination unreachable. I was able to send ICMP destination unreachable message to a designated host based on the IP address. However, I first sent a packet that was using a code field value that is known to Wireshark as "Port unreachable" (Figure 20). This packet passed through and was recognized with no errors on either end of the connection (Figure 21). The sniffer running on the receiving host also recognized it and as seen in Figure 22 it output the desired DTN specific message.

```
Source address: 172.20.109.81
Destination address: 172.20.105.248

Sent 28 byte packet to 172.20.105.248
Packet type: 3
Packet code: 0
```

Figure 20. Command line output of crafted ICMP type 3 packet

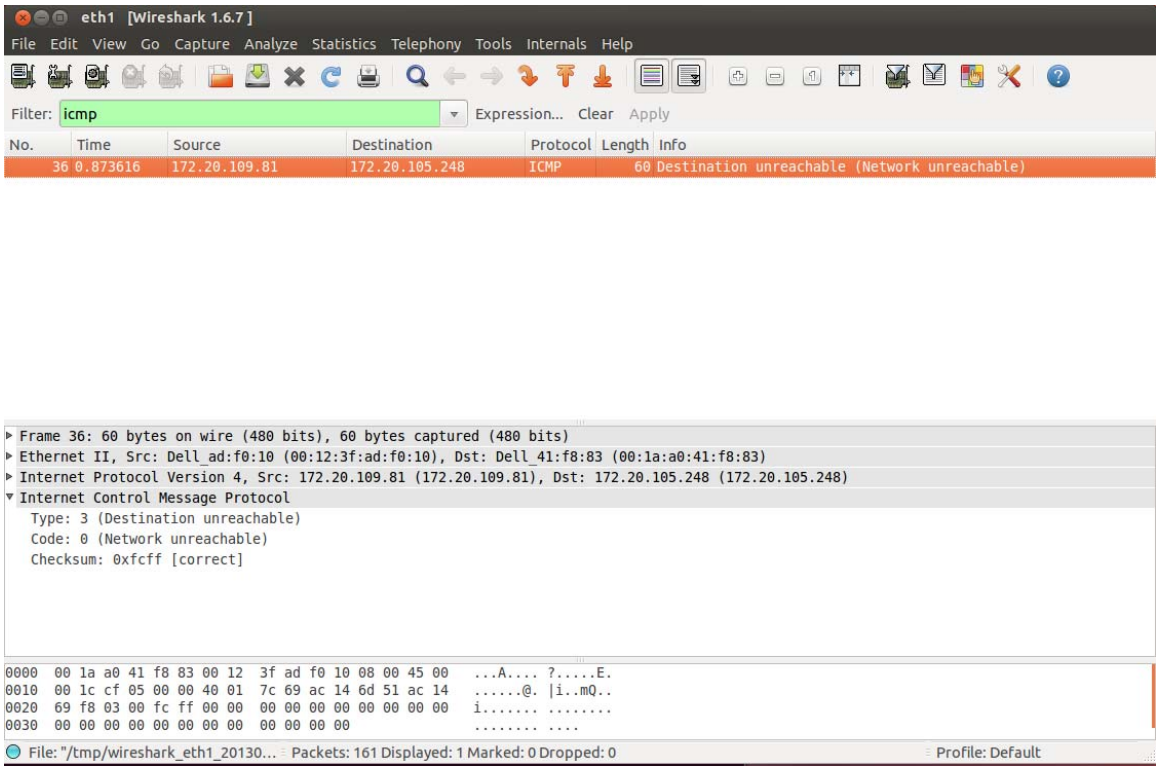

Figure 21. Wireshark Capture of ICMP type 3 known packet

```
Received ICMP packet of type 3. Success!
Packet has code value of 0
```

Figure 22. Sniffer output from ICMP type 3 code 0 packet

With a successful base test I could advance to a new code value and observe the packet. I inserted 20 as the code value since the recognized codes for ICMP Destination Unreachable packets are 0-15. Once the packet was inserted into the connection it was recognized clearly as ICMP destination unreachable and the code was "unknown" as expected in the Wireshark capture as seen in Figure 23. In the Sniffer application, it is recognized and the DTN specific message is output to the command line. This shows

that you can simulate ICMP destination unreachable packets with new codes to represent DTN unique messages.

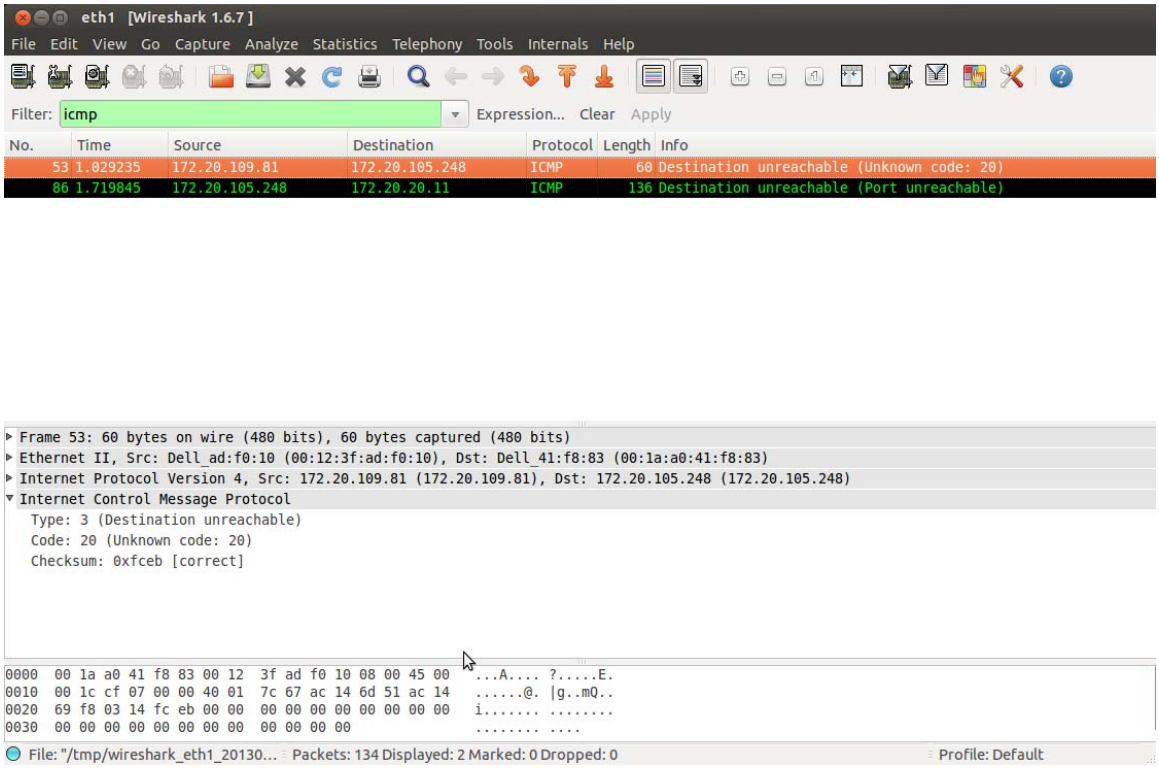

Figure 23. Wireshark capture with ICMP type 3 and code value 20

```
Received ICMP packet of type 3. Success!
Packet has code value of 20
Explicit Loss Notification

Received ICMP packet of type 3. Success!
Packet has code value of 21
Path re-established
```

Figure 24. Command line output from Sniffer for ICMP type 3 code 20 and 21

With the both methods of sending ICMP packets showing effective, it seems feasible to use ICMP packets to inform applications of the use of DTN transport for their traffic. Using a new ICMP type to make it specific to DTN is shown to work with Ubuntu hosts. The method of reusing a known ICMP message type with previously unused code values in the header also works according to my tests, and such messages should be less likely be filtered out by the host operating systems.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSION

The primary research presented in this thesis was the evaluation of whether the current release of the BBN SPINDLE Disruption Tolerant Networking software is deployment ready. Several tests were conducted to examine the installation and configuration process and associated documentation to determine the amount of effort required to accomplish it on a single node and for a network of nodes. As a secondary research item this thesis explored the task of altering an ICMP message to get DTN specific messages to applications. BBN makes it simple to a user with only two configuration files. With easy configuration and installation, deployment on multiple nodes is quickly performed. The major findings from this effort are summarized as follows.

- This thesis asked the question in the introduction about the stability of the code. With the functionality it currently provides it is my opinion that the BBN SPINDLE BPA is ready for more testing but not for deployment to the broader audience. The admin application issues and the subnet communication issues put the robust nature of the software into question. Currently testing is being done with the SPINDLE BPA software to bridge common chat software with the SPINDLE BPA DTN software to provide more robust chat agents to combat units. With successful chat application integration into DTN it could lead to other traffic applications being adjusted for DTN topologies.

- Tests were conducted on the first of the two scenarios detailed in the user guide. The results from the tests on scenario one and two from the user manual show the directions ran well but when a user deviates from this simple structure it is difficult to get the software to operate as it should. For testing in a lab this is viable and an expert user can accommodate for discrepancies seen however a novice admin in a combat situation may not be able to sit down and read through the user manual and read the configuration files to decipher the correct method to get the network up and operational. The valid configurations detailed in the user manual are not all functional in actual operation for the current release. This presents trouble when multiple network addressing is used for a single DTN. A commonly used tactic to break a larger network into several smaller subnets is difficult to do with the BBN SPINDLE software. Since the user manual describes the method to enter several multicast beacon groups into the network configuration it appears that the plan is to overcome this barrier and provide functionality for a DTN across the different subnets. However, testing shows that multiple network subnets do not communicate unless neighbors are identified at startup in the current release of the BBN SPINLDE software.

- This thesis concludes that BBN SPINLDE BPA software is a significant step forward in DTN network software for military units. It is operational in the basic form and is progressing toward a complete package that will operate in a combat environment. It will send and receive bundles while retaining the data during a disruption. It gives the network the ability to be disconnected completely from a gateway for significantly more time than your standard TCP/IP allows. It accommodates the same disruption internally between nodes making the disruption-tolerant nature valid for use. All the sent bundles arrived at the destination in sufficient time to be acceptable while still indulging a disruption to the connection. With the ability to adhere to multiple topologies through configuration files, a single node is very versatile.

- The reported errors from the first version provided testing to show the progress of the software toward an operational status. Testing showed that the BBN SPINDLE BPA software operates well when you use the basic applications to send and receive bundles through the basic topologies but more complicated admin applications are not ready for use. In release 20101221, the admin applications were hindered by not only custody issues but activation issues. The new release 20130109 makes the claim to have corrected this

error but my tests showed how this fix only partially works and still leaves applications lacking the robust versatility to be able to operate in wider environments outside the lab.

- In this research, tests were conducted on both static and dynamic routing topologies. The tests were limited by the small number of nodes but still show functionality and demonstrate the remaining errors (e.g., ***dtntraceroute*** and ***dtnreporter*** as documented in section IV.D) that need to be corrected before initiating a large install into combat units.

- Testing looked at the feasibility of a different ICMP type than normal to convey unique messages. The thesis then looked at how current existing messages could be used to get the unique codes to the applications. Testing code was written to show how easy it is to adjust a known and valid ICMP message and carry the unique code into the application. Using ICMP destination unreachable messages allows any node that determines the path to be disrupted to report back using an ICMP but with our changes it can have the DTN specific message.

With robust DTN software deployed to combat units in remote areas communication can remain fluid while the network changes dynamically. A DTN is a network of smaller networks. It is an overlay on top of special-purpose networks, including the Internet. DTNs support

interoperability of other networks by accommodating long disruptions and delays between and within those networks, and by translating between the communication protocols of those networks. In providing these functions, DTNs accommodate the mobility and limited power of evolving wireless communication devices. DTNs can accommodate many kinds of wireless technologies, including radio frequency (RF), ultra-wide band (UWB), free-space optical, and acoustic (sonar or ultrasonic) technologies. BBN SPINLDE BPA software could be the answer to this in the future.

For future evolutions the topology may be expanded to 8–10 DTN nodes connected through multiple links such as radio and wired that adapts to changing topology over time. Furthermore, several topologies may be manipulated during operation such as mesh to achieve a dynamic environment. Ultimately, the goal is to have DTN nodes that operate in a dynamic DTN topology where nodes may join or leave the network unexpectedly and are resistant to disruption attacks.

A. RECOMMENDATIONS FOR FUTURE WORK

For future testing of this type it would be helpful to perform these tests on other platforms to confirm the results are universal. Future work in this area is really boundless. DTNs are just starting to gain motivation towards everyday life applications. SPINDLE software testing is in the early stages of maturity with this release. Some future testing could be to use an develop an application that will monitor the disruptions and report compounding delays at each node allowing the node to be able to report disruptions to endpoints it has been aware

of previously. This will allow an algorithm to be written that can dynamically evaluate the discovered neighbors and expand or reduce the list as needed. With the dynamic entry and exit of nodes at such a rapid pace that DTNs are designed for. Nodes could also attempt discovery of the individual sensors on neighbors to gather data on node movement. Knowing where nodes have been and currently are can give the network a better idea of how to distribute files on the network. Knowing sensor data will assist in accessing content based searching rather than address based access to files. These are just a few research ideas that can be developed in the near future.

APPENDIX A

Excerpt from *bbn-bpa.ips*

```
# Specify static routes
# Format is: [type:options] nodelist
# Type is required, options are optional
#
# Valid Types are:
#   linear - connects nodes in a linear formation based on the nodes list
#
#            example: [linear] node1 node2 node3 node4 node5
#            node1 <-> node2 <-> node3 <-> node4 <-> node5
#
#   ring - like linear, but the ends are also connected
#
#          example: [ring] node1 node2 node3 node4 node5
#          node1 <-> node2 <-> node3 <-> node4 <-> node5 <-> node1
#
#   edge - first node on the nodes list connected to all others
#
#          example: [edge] node1 node2 node3 node4
#          node1 <-> node2, node1 <-> node3, node1 <-> node4
#
#   grid - NxN grid formation where P="total # nodes" and N=ceil(sqrt(P))
#          The grid is filled left to right and then top to bottom
#
#          example: [grid] node1 node2 node3 node4 node5 node6
#          node1 <-> node2 <-> node3
#            ^         ^         ^
#            |         |         |
#            V         V         V
#          node4 <-> node5 <-> node6
#
#   mesh - all nodes connected to all others
#
#          example: [mesh] node1 node2 node3
#          node1 <-> node2, node1 <-> node3, node2 <-> node3
#
#   mesh - all nodes connected to all others
#
#          example: [mesh] node1 node2 node3
#          node1 <-> node2, node1 <-> node3, node2 <-> node3
#
# By default, the topology will be set up using static multihop routes
# installed. This can be changed with the options below:
#
# The following options are mutually exclusive of each other:
#
#   path           Notes a routing path which we expect routers to take care of.
#                  The purpose of this is to create mixed-mode setups where a
#                  portion of the network uses static routes and another portion
#                  uses ad-hoc routers
#   no_route       Tells the parser to not install any sort of multi-hop routes
#
# The following options are mutually exclusive of each other:
#
#   nd_unicast     Tells the parser to install unicast beacons
#                  one-way links (i.e., node1->node2)
#   no_nd          Causes a permanent link to be created at startup. Note, when
#                  using the TCP CLA, if one side tears down the link, the link
#                  will be taken down on both nodes
#
# The following options may be added with any of the above
#
#   unidirectional Tells the parser all specified links should be created
```

```
#
#
# Routes may be specified across multiple lines
#
# NOTE: Current functionality only allows for topologies to include nodes
#       that belong to the same multicast beacon group as specified by the
#       udpnd_beacon_groups configuration below.
#
# example:
#   topology = {
#      [linear] node1 node2 node3 node4
#      [linear:nd_unicast] node5 node6 node7 node8 node9
#      [linear:unidirectional] node10 node11 node12 node13 node14
#      [edge:path:unidirectional] node15 node16 node17
#                                 node18 node19
#   }

topology = {
}
```

APPENDIX B

Excerpt from *bbn-bpa.config*

```
# (required) List of CLAs we're using
#
# allowed arguments: udp, tcp, pudp, norm
CLAS = udp

# (optional) List of Routers we're using
#
# allowed arguments: dr, flood, tmrg, cba-sflood, gran
ROUTERS = dr

# (required) List of Neighbor Discovery Adapters we're using. When we're
# running with UDP based Neighbor Discovery, the udpnd_beacondaddress in the
# "udpnd_beacon_groups" settings in .ips file.
#
# UDPND will send beacons on all ethernet interfaces used by this node as
# specified in the .ips file. If we're running with DR, UDPND will be
# automatically enabled.
#
# allowed arguments: udp
ND_ADAPTERS = udp
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

```
1   /*
2   * crafter.c
3   *
4   *  Created on:   March 19, 2013
5   *  Author:       Todd Sehl
6   *
7   *  NOTES:
8   *    Original base code for ICMP mesage creation from
9   *    pinger.c code found on a programming forum board:
10  *    "http://cboard.cprogramming.com/networking-device-
11  *    communication/41635-ping-program.html"
12  *
13  *    Changes were made to the application to allow user
14  *    the options to change the type and code values on
15  *    the command line for the ICMP packets generated.
16  *    Crafter.c allows user to input a unique type and
17  *    code field values.
18  *
19  */
20
21
22  #include <stdio.h>
23  #include <stdlib.h>
24  #include <sys/types.h>
25  #include <sys/socket.h>
26  #include <netinet/in.h>
27  #include <arpa/inet.h>
28  #include <netdb.h>
29  #include <linux/ip.h>
30  #include <linux/icmp.h>
31  #include <string.h>
32  #include <unistd.h>
33
34
35  unsigned short in_cksum(unsigned short *, int);
36  void usage();
37  char* getip();
38  char* toip(char*);
39
40  int main(int argc, char* argv[])
41  {
42  struct iphdr* ip;
43  struct icmphdr* icmp;
44  struct sockaddr_in connection;
45  char* packet;
```

```
46    char* buffer;
47    int sockfd;
48    int optval;
49    char **av = argv;
50    char src_addr[20];
51    char dst_addr[20];
52    int ICMP_TYPE_VAL = 192;
53    int ICMP_CODE_VAL = 0;
54    int src_set = 0;
55
56    if (getuid() != 0)
57    {
58    fprintf(stderr, "%s: root privileges needed\n," *(argv + 0));
59    exit(EXIT_FAILURE);
60    }
61
62    /* there are too many options on the command line */
63    if(argc == 1)
64    {
65    usage();
66    exit(EXIT_FAILURE);
67    }
68
69    /* there are too many options on the command line */
70    if(argc > 8)
71    {
72    printf("You have provided too many arguments for this program.");
73    usage();
74    exit(EXIT_FAILURE);
75    }
76
77    if (*av == '-h')
78    {
79    usage();
80    exit(EXIT_FAILURE);
81    }
82
83
84    argc--, av++;
85    //Parse out arguments from command line
86    if (argc == 1)
87    {
88    /*
89    *      only one argument provided
90    *      assume it is the destination server
91    *      source address is local host
92    */
93    strncpy(dst_addr, *(av), 15);
94    strncpy(src_addr, getip(), 15);
```

```
95    }
96    else if(argc > 1)
97    {
98    strncpy(dst_addr, *(av), 15);
99    argc--, av++;
100   while (argc > 0 && *av[0] == '-')
101   {
102   while (*++av[0])
103   {
104   switch (*av[0])
105   {
106   case 's':
107   src_set = 1;
108   av++;
109   strncpy(src_addr, *(av), 15);
110   printf("source\n");
111   break;
112   case 't':
113   av++;
114   ICMP_TYPE_VAL = atoi(av[0]);
115   printf("type\n");
116   break;
117   case 'c':
118   av++;
119   ICMP_CODE_VAL = atoi(av[0]);
120   printf("code\n");
121   break;
122   case 'h':
123   usage();
124   exit(EXIT_FAILURE);
125   }
126   argc--;
127   if (av[1] != NULL)
128   {
129   av++;
130   }
131   }
132   }
133   }
134
135   if (src_set == 0)
136   {
137   strncpy(src_addr, getip(), 15);
138   }
139
140   strncpy(dst_addr, toip(dst_addr), 20);
141   strncpy(src_addr, toip(src_addr), 20);
142   printf("Source address: %s\n," src_addr);
143   printf("Destination address: %s\n," dst_addr);
```

```
144
145   /*
146   * allocate all necessary memory
147   */
148   packet = malloc(sizeof(struct iphdr) + sizeof(struct icmphdr));
149   buffer = malloc(sizeof(struct iphdr) + sizeof(struct icmphdr));
150   /*************************************************************/
151
152   ip = (struct iphdr*) packet;
153   icmp = (struct icmphdr*) (packet + sizeof(struct iphdr));
154
155   /*
156   *  here the ip packet is set up
157   */
158   ip->ihl          = 5;
159   ip->version      = 4;
160   ip->tos     = 0;
161   ip->tot_len = sizeof(struct iphdr) + sizeof(struct icmphdr);
162   ip->id= htons(0);
163   ip->frag_off= 0;
164   ip->ttl     = 64;
165   ip->protocol= IPPROTO_ICMP;
166   ip->saddr   = inet_addr(src_addr);
167   ip->daddr   = inet_addr(dst_addr);
168   ip->check   =   in_cksum((unsigned   short   *)ip,   sizeof(struct
      iphdr));
169
170   if ((sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)) == -1)
171   {
172   perror("socket");
173   exit(EXIT_FAILURE);
174   }
175
176   /*
177   *  IP_HDRINCL must be set on the socket so that
178   *  the kernel does not attempt to automatically add
179   *  a default ip header to the packet
180   */
181
182   setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &optval, sizeof(int));
183
184   /*
185   *  here the icmp packet is created
186   *  also the ip checksum is generated
187   */
188   icmp-> type          = ICMP_TYPE_VAL;
189   icmp-> code          = ICMP_CODE_VAL;
190   icmp-> checksum      = in_cksum((unsigned short *)icmp,
      sizeof(struct icmphdr));
191
192   connection.sin_family = AF_INET;
193   connection.sin_addr.s_addr = inet_addr(dst_addr);
```

```
194
195   /*
196   *  now the packet is sent
197   */
198   sendto(sockfd, packet, ip->tot_len, 0, (struct sockaddr
      *)&connection, sizeof(struct sockaddr));
199   printf("\nSent %d byte packet to %s\nPacket type: %d\nPacket code:
      %d\n\n," ip->tot_len, dst_addr,icmp->type, icmp->code);
200
201   free(packet);
202   free(buffer);
203   close(sockfd);
204   return 0;
205 }
206
207 void usage()
208 {
209   fprintf(stderr, "\nUsage: crafter [destination] <-s [source]> <-t
      [type]> <-c [code]>\n");
210   fprintf(stderr, "OPTIONS:\t-s\tsource IP address\n\t\t-
      t\tICMP message type value\n\t\t-c\tICMP message code
      value\n\n");
211   fprintf(stderr, "Destination must be provided.\n");
212   fprintf(stderr, "Source, type, and code are optional.\n");
213   fprintf(stderr, "Defaults:\tsource = localhost \n\t\ttype = 192
      \n\t\tcode = 0\n\n");
214   fprintf(stderr, "Example: crafter 192.120.20.210 -t 3 -c
      22\n\n\n");
215 }
216
217 char* getip()
218 {
219   char buffer[256];
220   struct hostent* h;
221
222   gethostname(buffer, 256);
223   h = gethostbyname(buffer);
224
225   return inet_ntoa(*(struct in_addr *)h->h_addr);
226
227 }
228
229 /*
230  * return the ip address if host provided by DNS name
231  */
232 char* toip(char* address)
233 {
234   struct hostent* h;
235   h = gethostbyname(address);
```

```
236   return inet_ntoa(*(struct in_addr *)h->h_addr);
237 }
238
239  /*
240   * in_cksum --
241   * Checksum routine for Internet Protocol
242   * family headers (C Version)
243   */
244 unsigned short in_cksum(unsigned short *addr, int len)
245 {
246   register int sum = 0;
247   u_short answer = 0;
248   register u_short *w = addr;
249   register int nleft = len;
250   /*
251   * Our algorithm is simple, using a 32 bit accumulator (sum), we
      add
252   * sequential 16 bit words to it, and at the end, fold back all the
253   * carry bits from the top 16 bits into the lower 16 bits.
254   */
255   while (nleft > 1)
256   {
257   sum += *w++;
258   nleft -= 2;
259   }
260   /* mop up an odd byte, if necessary */
261   if (nleft == 1)
262   {
263   *(u_char *) (&answer) = *(u_char *) w;
264   sum += answer;
265   }
266   /* add back carry outs from top 16 bits to low 16 bits */
267   sum = (sum >> 16) + (sum & 0xffff);   /* add hi 16 to low 16 */
268   sum += (sum >> 16);                    /* add carry */
269   answer = ~sum;                         /* truncate to 16 bits */
270   return (answer);
271 }
```

APPENDIX D

```
1   /*
2   * sniffer.c
3   *
4   * Date: March 19, 2013
5   * Update Author: Todd Sehl
6   * Original Author: Unkown
7   * Location: http://www.binarytides.com/packet-sniffer-
8   * code-in-c-using-linux-sockets-bsd/
9   *
10  * NOTE:
11  * This code was found on a website for learning to
12  * write a packet sniffer. It performs the task of showing
13  * the ICMP arrival. I did not write it. I only added code to
14  * print to the screen when the teh DTN unique ICMP packets
15  * are captured.
16  *
17  */
18
19
20  #include<stdio.h> //For standard things
21  #include<stdlib.h>//malloc
22  #include<string.h>//memset
23  #include<netinet/ip_icmp.h> //Provides declarations for icmp header
24  #include<netinet/udp.h> //Provides declarations for udp header
25  #include<netinet/tcp.h> //Provides declarations for tcp header
26  #include<netinet/ip.h>    //Provides declarations for ip header
27  #include<sys/socket.h>
28  #include<arpa/inet.h>
29
30  void ProcessPacket(unsigned char* , int);
31  void print_ip_header(unsigned char* , int);
32  void print_tcp_packet(unsigned char* , int);
33  void print_udp_packet(unsigned char * , int);
34  void print_icmp_packet(unsigned char* , int);
35  void PrintData (unsigned char* , int);
36
37  int sock_raw;
38  FILE *logfile;
39  int tcp=0,udp=0,icmp=0,others=0,igmp=0,total=0,i,j;
40  struct sockaddr_in source,dest;
41
42  int main()
43  {
44  int saddr_size , data_size;
45  struct sockaddr saddr;
46  struct in_addr in;
47
48  unsigned char *buffer = (unsigned char *)malloc(65536); //Its Big!
49
50  logfile=fopen("log.txt,""w");
```

```
51  if(logfile==NULL) printf("Unable to create file.");
52  printf("Starting...\n");
53  //Create a raw socket that shall sniff
54  sock_raw = socket(AF_INET , SOCK_RAW , IPPROTO_ICMP);
55  if(sock_raw < 0)
56  {
57  printf("Socket Error\n");
58  return 1;
59  }
60  while(1)
61  {
62  saddr_size = sizeof saddr;
63  //Receive a packet
64  data_size = recvfrom(sock_raw , buffer , 65536 , 0 , &saddr ,
      &saddr_size);
65  if(data_size <0 )
66  {
67  printf("Recvfrom error , failed to get packets\n");
68  return 1;
69  }
70  //Now process the packet
71  ProcessPacket(buffer , data_size);
72  }
73  close(sock_raw);
74  printf("Finished");
75  return 0;
76  }
77
78  void ProcessPacket(unsigned char* buffer, int size)
79  {
80  //Get the IP Header part of this packet
81  struct iphdr *iph = (struct iphdr*)buffer;
82  ++total;
83  switch (iph->protocol) //Check the Protocol and do accordingly...
84  {
85  case 1:     //ICMP Protocol
86  print_icmp_packet(buffer, size);
87  ++icmp;
88  //PrintIcmpPacket(Buffer,Size);
89  break;
90
91  case 2:     //IGMP Protocol
92  ++igmp;
93  break;
94
95  case 6:     //TCP Protocol
96  ++tcp;
97  print_tcp_packet(buffer , size);
98  break;
99
100 case 17: //UDP Protocol
101 ++udp;
102 print_udp_packet(buffer , size);
103 break;
```

```
104
105 default: //Some Other Protocol like ARP etc.
106 ++others;
107 break;
108 }
109 printf("TCP : %d   UDP : %d  ICMP : %d  IGMP : %d  Others    :    %d
      Total : %d\r,"tcp,udp,icmp,igmp,others,total);
110 }
111
112 void print_ip_header(unsigned char* Buffer, int Size)
113 {
114 unsigned short iphdrlen;
115
116 struct iphdr *iph = (struct iphdr *)Buffer;
117 iphdrlen =iph->ihl*4;
118
119 memset(&source, 0, sizeof(source));
120 source.sin_addr.s_addr = iph->saddr;
121
122 memset(&dest, 0, sizeof(dest));
123 dest.sin_addr.s_addr = iph->daddr;
124
125 fprintf(logfile,"\n");
126 fprintf(logfile,"IP Header\n");
127 fprintf(logfile,"   |-IP Version      :   %d\n,"(unsigned   int)iph-
      >version);
128 fprintf(logfile,"   |-IP Header Length:    %d    DWORDS    or    %d
      Bytes\n,"(unsigned int)iph->ihl,((unsigned int)(iph->ihl))*4);
129 fprintf(logfile," |-Type Of Service   :  %d\n,"(unsigned  int)iph-
      >tos);
130 fprintf(logfile," |-IP Total Length   : %d  Bytes(Size          of
      Packet)\n,"ntohs(iph->tot_len));
131 fprintf(logfile,"   |-Identification  : %d\n,"ntohs(iph->id));
132 //fprintf(logfile,"   |-Reserved ZERO Field :
      %d\n,"(unsigned int)iphdr->ip_reserved_zero);
133 //fprintf(logfile,"   |-Dont Fragment Field :
      %d\n,"(unsigned int)iphdr->ip_dont_fragment);
134 //fprintf(logfile,"   |-More Fragment Field :
      %d\n,"(unsigned int)iphdr->ip_more_fragment);
135 fprintf(logfile," |-TTL              :   %d\n,"(unsigned   int)iph-
      >ttl);
136 fprintf(logfile," |-Protocol         :   %d\n,"(unsigned   int)iph-
      >protocol);
137 fprintf(logfile," |-Checksum         : %d\n,"ntohs(iph->check));
138 fprintf(logfile," |-Source IP        :
      %s\n,"inet_ntoa(source.sin_addr));
139 fprintf(logfile,"   |-Destination IP :
      %s\n,"inet_ntoa(dest.sin_addr));
140 }
141
142 void print_tcp_packet(unsigned char* Buffer, int Size)
143 {
144 unsigned short iphdrlen;
145
```

```
146 struct iphdr *iph = (struct iphdr *)Buffer;
147 iphdrlen = iph->ihl*4;
148
149 struct tcphdr *tcph=(struct tcphdr*)(Buffer + iphdrlen);
150
151 fprintf(logfile,"\n\n**********************TCP
      Packet************************\n");
152
153 print_ip_header(Buffer,Size);
154
155 fprintf(logfile,"\n");
156 fprintf(logfile,"TCP Header\n");
157 fprintf(logfile,"   |-Source Port    : %u\n,"ntohs(tcph->source));
158 fprintf(logfile,"   |-Destination Port : %u\n,"ntohs(tcph->dest));
159 fprintf(logfile,"   |-Sequence Number    : %u\n,"ntohl(tcph->seq));
160 fprintf(logfile,"   |-Acknowledge Number : %u\n,"ntohl(tcph-
      >ack_seq));
161 fprintf(logfile,"   |-Header Length      : %d DWORDS or %d BYTES\n"
      ,(unsigned int)tcph->doff,(unsigned int)tcph->doff*4);
162 //fprintf(logfile,"   |-CWR Flag : %d\n,"(unsigned int)tcph->cwr);
163 //fprintf(logfile,"   |-ECN Flag : %d\n,"(unsigned int)tcph->ece);
164 fprintf(logfile,"   |-Urgent Flag          : %d\n,"(unsigned
      int)tcph->urg);
165 fprintf(logfile,"   |-Acknowledgement Flag : %d\n,"(unsigned
      int)tcph->ack);
166 fprintf(logfile,"   |-Push Flag            : %d\n,"(unsigned
      int)tcph->psh);
167 fprintf(logfile,"   |-Reset Flag           : %d\n,"(unsigned
      int)tcph->rst);
168 fprintf(logfile,"   |-Synchronise Flag     : %d\n,"(unsigned
      int)tcph->syn);
169 fprintf(logfile,"   |-Finish Flag          : %d\n,"(unsigned
      int)tcph->fin);
170 fprintf(logfile,"   |-Window         : %d\n,"ntohs(tcph->window));
171 fprintf(logfile,"   |-Checksum       : %d\n,"ntohs(tcph->check));
172 fprintf(logfile,"   |-Urgent Pointer : %d\n,"tcph->urg_ptr);
173 fprintf(logfile,"\n");
174 fprintf(logfile,"                        DATA Dump                         ");
175 fprintf(logfile,"\n");
176
177 fprintf(logfile,"IP Header\n");
178 PrintData(Buffer,iphdrlen);
179
180 fprintf(logfile,"TCP Header\n");
181 PrintData(Buffer+iphdrlen,tcph->doff*4);
182
183 fprintf(logfile,"Data Payload\n");
184 PrintData(Buffer + iphdrlen + tcph->doff*4 , (Size - tcph->doff*4-
      iph->ihl*4) );
185
186
       fprintf(logfile,"\n###########################################
      ############");
187 }
188
```

```
189 void print_udp_packet(unsigned char *Buffer , int Size)
190 {
191
192  unsigned short iphdrlen;
193
194  struct iphdr *iph = (struct iphdr *)Buffer;
195  iphdrlen = iph->ihl*4;
196
197  struct udphdr *udph = (struct udphdr*)(Buffer + iphdrlen);
198
199  fprintf(logfile,"\n\n**********************UDP
       Packet************************\n");
200
201  print_ip_header(Buffer,Size);
202
203  fprintf(logfile,"\nUDP Header\n");
204    fprintf(logfile," |-Source Port    :    %d\n"    ,    ntohs(udph-
       >source));
205    fprintf(logfile," |-Destination Port : %d\n" , ntohs(udph->dest));
206    fprintf(logfile," |-UDP Length     : %d\n" , ntohs(udph->len));
207    fprintf(logfile," |-UDP Checksum   : %d\n" , ntohs(udph->check));
208
209    fprintf(logfile,"\n");
210    fprintf(logfile,"IP Header\n");
211    PrintData(Buffer , iphdrlen);
212
213    fprintf(logfile,"UDP Header\n");
214    PrintData(Buffer+iphdrlen , sizeof udph);
215
216    fprintf(logfile,"Data Payload\n");
217    PrintData(Buffer + iphdrlen + sizeof udph ,( Size - sizeof udph -
       iph->ihl * 4 ));
218
219
       fprintf(logfile,"\n###############################################
       ##############");
220 }
221
222 void print_icmp_packet(unsigned char* Buffer , int Size)
223 {
224    unsigned short iphdrlen;
225    unsigned short icmp_code;
226
227    struct iphdr *iph = (struct iphdr *)Buffer;
228    iphdrlen = iph->ihl*4;
229
230    struct icmphdr *icmph = (struct icmphdr *)(Buffer + iphdrlen);
231
232
       fprintf(logfile,"\n\n**************
       *********ICMP
       Packet************************\n")
       ;
233
234    print_ip_header(Buffer , Size);
```

```
235
236   fprintf(logfile,"\n");
237
238   fprintf(logfile,"ICMP Header\n");
239   fprintf(logfile," |-Type : %d,"(unsigned int)(icmph->type));
240
241   if((unsigned int)(icmph->type) == 11)
242   fprintf(logfile,"  (TTL Expired)\n");
243   else if((unsigned int)(icmph->type) == ICMP_ECHOREPLY)
244   fprintf(logfile,"  (ICMP Echo Reply)\n");
245   fprintf(logfile," |-Code : %d\n,"(unsigned int)(icmph->code));
246   fprintf(logfile," |-Checksum : %d\n,"ntohs(icmph->checksum));
247
248   icmp_code = icmph->code;
249   //Added code to print to screen if the type is 192
250   if((unsigned int)(icmph->type) == 192)
251   {
252   printf("Recieved ICMP packet of type 192. Success!\nPacket has
      code value of %d\n," icmp_code);
253   switch(icmp_code)
254   {
255   case 0:
256   printf("Explicit Loss Notification \n");
257   break;
258
259   case 1:
260   printf("Path re-established \n");
261   break;
262
263   case 2:
264   printf("Explicit Delay Notification < 10 s \n");
265   break;
266
267   case 3:
268   printf("Explicit Delay Notification < 1 min \n");
269   break;
270
271   case 4:
272   printf("Explicit Delay Notification < 10 min \n");
273   break;
274
275   case 5:
276   printf("Explicit Delay Notification < 1 hr \n");
277   break;
278
279   case 6:
280   printf("Explicit Delay Notification unknown \n");
281   break;
282   }
283   }
284
285
286
287
```

```
288   //Added code to print messages to screen from Destination
      Unreachable
289   if(((unsigned int)(icmph->type) == 3) && ((unsigned int)(icmph-
      >code) >= 20))
290   {
291   printf("Received ICMP packet of type 3. Success!\nPacket has code
      value of %d\n," icmp_code);
292
293   switch(icmp_code)
294   {
295   case 20:
296   printf("Explicit Loss Notification \n");
297   break;
298
299   case 21:
300   printf("Path re-established \n");
301   break;
302
303   case 22:
304   printf("Explicit Delay Notification < 10 s \n");
305   break;
306
307   case 23:
308   printf("Explicit Delay Notification < 1 min \n");
309   break;
310
311   case 24:
312   printf("Explicit Delay Notification < 10 min \n");
313   break;
314
315   case 25:
316   printf("Explicit Delay Notification < 1 hr \n");
317   break;
318
319   case 26:
320   printf("Explicit Delay Notification unknown \n");
321   break;
322   }
323   }
324   //fprintf(logfile," |-ID      : %d\n,"ntohs(icmph->id));
325   //fprintf(logfile," |-Sequence : %d\n,"ntohs(icmph->sequence));
326   fprintf(logfile,"\n");
327
328   fprintf(logfile,"IP Header\n");
329   PrintData(Buffer,iphdrlen);
330
331   fprintf(logfile,"UDP Header\n");
332   PrintData(Buffer + iphdrlen , sizeof icmph);
333
334   fprintf(logfile,"Data Payload\n");
335   PrintData(Buffer + iphdrlen + sizeof icmph , (Size - sizeof icmph
      - iph->ihl * 4));
336
```

```
337
      fprintf(logfile,"\n##############################################
      #############");
338 }
339
340 void PrintData (unsigned char* data , int Size)
341 {
342
343   for(i=0 ; i < Size ; i++)
344   {
345   if( i!=0 && i%16==0)   //if   one   line   of   hex   printing   is
      complete...
346   {
347   fprintf(logfile,"          ");
348   for(j=i-16 ; j<i ; j++)
349   {
350   if(data[j]>=32 && data[j]<=128)
351   fprintf(logfile,"%c,"(unsigned  char)data[j]);  //if  its  a
      number or alphabet
352
353   else fprintf(logfile,."");  //otherwise print a dot
354   }
355   fprintf(logfile,"\n");
356   }
357
358   if(i%16==0) fprintf(logfile," ");
359   fprintf(logfile," %02X,"(unsigned int)data[i]);
360
361   if( i==Size-1)   //print the last spaces
362   {
363   for(j=0;j<15-i%16;j++) fprintf(logfile,"   "); //extra spaces
364
365   fprintf(logfile,"          ");
366
367   for(j=i-i%16 ; j<=i ; j++)
368   {
369   if(data[j]>=32              &&              data[j]<=128)
      fprintf(logfile,"%c,"(unsigned char)data[j]);
370   else fprintf(logfile,."");
371   }
372   fprintf(logfile,"\n");
373   }
374   }
375 }
```

LIST OF REFERENCES

[1] K. Fall, "A delay-tolerant network architecture for challenged Internets," ***SIGCOMM'03,*** pp. 27–34, 2003.

[2] S. Burleigh, V. Cerf, R. Durst, K. Fall, A. Hooke and K. Scott, "The interplanetary Internet: A communications infrastructure for Mars exploration," ***Acta Astronautica,*** vol. 53, no. 4–10, pp. 365–373, 2003.

[3] F. Warthman, "Delay- and Disruption-Tolerant Networks A Primer," Interplanetary Internet Special Interest Group, 2012.

[4] D. T. N. R. Group, "Compiling DTN2," Internet Research Task Force (IRTF), 2013. [Online]. Available: http://www.dtnrg.org/docs/code/DTN2/doc/manual/compiling.html. [Accessed 10 March 2013].

[5] BBN Technologies, ***Latency-Aware Information Access with User-Directed Fetch Behaviour for Weakly-Connected Mobile Wireless Clients,*** Cambridge, MA: Internetwork Research, 2002.

[6] R. Krishnan, P. Basu, J. M. Mikkelson, C. Small, R. Ramanathan, D. W. Brown, J. R. Burgess, A. L. Caro, M. Condell, N. C. Goffee, R. R. Hain, R. E. Hansen, C. E. Jones, V. Kawadia, D. P. Mankins, B. I. Schwartz, W. T. Strayer, J. W. Ward, D. P. Wiggins and S. H. Polit, ***The SPINDLE Disruption-Tolerant Networking System,*** Cambridge, MA: BBN Technologies, 2007.

[7] Information Sciences Institute University of Southern California, "Transmission Control Protocol DARPA Internet Program Protocol Specification," Information Sciences Institute University of Southern California, Marina del Rey, 1981.

[8] K. Scott and S. Burleigh, "Bundle Protocol Specification (RFC 5050)," The IETF Trust, 2007.

[9] BBN Technologies Corp, ***SPINDLE DTN API,*** Cambridge, MA: BBN Technologies Corp, 2009.

[10] Raytheon BBN Technologies Corporation, ***Software User Manual for the USMC Disruption Tolerant Networking,*** Cambridge, 2012.

[11] Ekiga.org, "Ekiga softphone features," Ekiga.org, [Online]. Available: http://www.ekiga.org/ekiga-softphone-features. [Accessed 2 February 2013].

[12] J. Rohrer and G. Xie, ***An application transparent approach to integrating ip and disruption tolerant networks,*** Monterey: Unpublished manuscript, 2012.

[13] Ekiga, ***Open Phone Abstraction Library Overview,*** 2010.

[14] D. Smithies, "OPAL - Open Phone Abstraction Library," 28 February 2005. [Online]. Available: http://www.opalvoip.org/docs/opal-v3_10/. [Accessed 10 February 2013].

[15] C. Southeren, "Portable Tools Library," Equivalence Pty Ltd, 17 May 2004. [Online]. Available: http://www.opalvoip.org/docs/ptlib-v2_10/. [Accessed 12 February 2013].

[16] J. Postel, "Internet Control Message Protocol, DARPA Internet Program Protocol Specification (RFC792)," Network Working Group, September 1981.

[17] S. Walton, Linux® Socket Programming, Indianapolis: Sams, 2001.

[18] Cprogramming.com, "cboard.programming.com - ping program," vBulletin Solutions, Inc, 2013. [Online]. Available: http://cboard.cprogramming.com/networking-device-communication/41635-ping-program.html. [Accessed 22 January 2013].

[19] S. Burleigh, V. Cerf, L. Torgerson, R. Durst, K. Scott, K. Fall and H. Weiss, ***Delay-tolerant network architecture,*** The IETF Trust, 2007.

[20] V. Cerf, Y. Dalal and C. Sunshine, "Specification of Internet Transmission Control Program," Network Working Group, 1974.

[21] J. N. Hoover, “DoD pushes military’s mobile strategy forward,” InformationWeek Government, 26 October 2012. [Online]. Available: http://www.informationweek.com/government/mobile/dod-pushes-militarys-mobile-strategy-for/240010603. [Accessed 20 January 2013].

[22] OPALVoIP.org, “OPALVoIPWiki FAQs,” OPALVoIP.org, [Online]. Available: http://www.opalvoip.org/pmwiki/pmwiki.php?n=Main.FAQ. [Accessed 8 February 2013].

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Systems Command